WHOI-90-22

2

Woods Hole Oceanographic Institution

VOICE – A Spectrogram Computer Display Package

by

Ann Martin, Josko A. Catipovic, Kurt Fristrup, and Peter L. Tyack

June 1990

Technical Report

Funding was provided by the Office of Naval Research through
Grant Nos. N00014-88-K-0273 and N00014-87-K-0236,
the National Institutes of Health through Grant No. 1 R29 NS25290,
and the Andrew W. Mellon Foundation.

Approved for public release; distribution unlimited.

WHOI-90-22

VOICE – A Spectrogram Computer Display Package

by

Ann Martin, Josko A. Catipovic, Kurt Fristrup, and Peter L. Tyack

Woods Hole Oceanographic Institution
Woods Hole, Massachusetts 02543

July 1990

Technical Report

Funding was provided by the Office of Naval Research through
Grant Nos. N00014-88-K-0273 and N00014-87-K-0236,
the National Institutes of Health through Grant No. 1 R29 NS25290,
and the Andrew W. Mellon Foundation.

Approved for publication; distribution unlimited.

Approved for Distribution:

Albert J. Williams 3rd

Albert J. Williams 3rd, Chairman
Department of Applied Ocean Physics and Engineering

Abstract

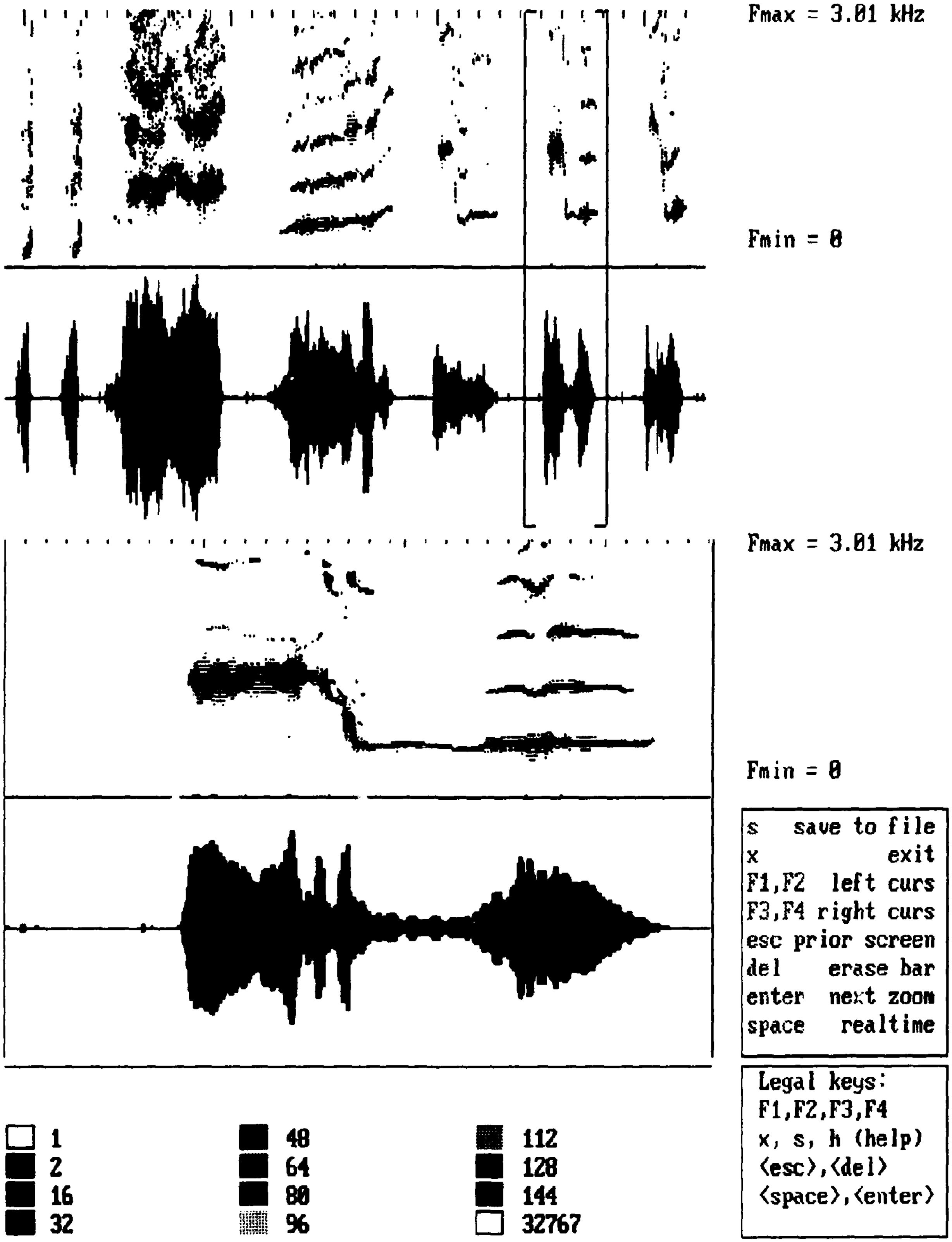
A real-time spectrogram instrument has been developed to provide an inexpensive and field-portable instrument for the analysis of animal sounds. The instrument integrates a computer graphics display package with a PC-AT computer equipped with an A/D board and a digital signal processing board. It provides a real-time spectrogram display of frequencies up to 50kHz in a variety of modes: a running display, a signal halted on screen, successive expanded views of the signal. The signal amplitude may also be displayed. Portions of the scrolled data may be saved to disk file for future viewing, or as part of a database collection. The screen display may be manipulated to adapt to special needs. Program source listings are included in the text. Keywords: Computer interface, spectrographs, Computer programming. (EDR) —

Acknowledgments

The development of the VOICE spectrographic display package has been a joint project between the AEL signal processing group of the AOPE Department and the Marine Animal Bioacoustics Laboratory of the Biology Department. Funding for the work has been from the Office of Naval Research under contracts N00014-88-K-0273 (Watkins/Fristrup) and N00014-87-K-0236 (Tyack), and National Institutes of Health grant 1 R29 NS25290 (Tyack). Support also was from The Andrew W. Mellon Foundation (Catipovic/Martin). The detailed description of the software development has been primarily by Ann Martin.

Table of Contents

Tables



The upper panel illustrates a section of humpback whale song in the memory display mode of VOICE. Cursors bracket the section that was expanded to produce the display in the lower panel. The lower panel also illustrates the help and message windows.

Introduction

A real-time spectrogram instrument has been developed to provide an inexpensive and field-portable instrument for the analysis of marine animal sounds. Named VOICE, the computer graphics display package is a combination of software and hardware components. The software has been designed so that a user with minimal computer experience can integrate this tool with a suitable application.

VOICE displays both spectrograms and waveform displays in real time on a computer screen. Sounds prerecorded on an analog audio tape can be directed simultaneously into both an amplifier and the VOICE computer. A microphone or hydrophone connected to the VOICE computer can also be used to display live animal calls on the screen. The ability to see the spectrograms of the sounds at the same time that they are heard greatly facilitates the identification of patterns that might go unnoticed when scanning depends solely on the human ear.

The developed tool provides a real-time spectrogram and waveform display, on-line save buffer editing and disk storage. Some of the instrument's capabilities are listed below:

> Continuously digitize an analog channel at an aggregate acquisition rate of 100 k samples/sec.
>
> Compute and display in real time a spectrogram and its envelope waveform with a clipping indicator at screen scroll rates up to 7 sec per monitor width.
>
> Halt the screen so that a spectrogram can be viewed as long as desired.
>
> Delimit a spectrogram signal with cursors for successive expansions of events too detailed to be examined in real time.
>
> Customize the screen display by controlling the sample frequency, setting the number of points per transform, setting the interval between time ticks marking the elapsed time for the data to scroll across the screen, modifying the levels at which colors change, modifying the color spectrum.
>
> Save delimited segments of digitized data to disk. The maximum save buffer length is 384 kbytes.
>
> Replay spectrogram data which has been saved during an earlier session.

The hardware unit consists of a PC AT with an EGA display and hard disk, an analog-to-digital (A/D) converter board and a digital signal processing board; these added components cost about $2500. In addition, a bandpass filter/preamp device such as "Frequency

Devices 9002" (about $2800) can be used between the audio tape and the A/D system to control input gain and to prevent aliasing. The cost is low compared to commercial spectrogram machines. The package was tested on a number of portable PC's (such as the NEC PowerMate Portable), which are well suited for field instrument use. This allows powerful signal processing and data acquisition capabilities to be brought into the field at modest cost.

The instrument development was motivated directly by the need of WHOI biologists to scan extensive analog data sets of marine mammal vocalizations, where the goal is to extract digital representations of exemplary marine mammal calls. It is difficult for many researchers to afford the spectrogram analysis workstations that are available commercially. In any case, such instruments are not suitable for field use, particularly in the remote locations frequented by WHOI researchers. The developed package allows for efficient data analysis and acquisition capabilities by Institution researchers at a reasonable cost, and thus contributes to their overall observational capabilities.

System Overview

The main hardware element is a PC-AT compatible personal computer with either an 80286 or 80386 CPU. An 8 MHz machine is adequate, although faster clock speeds and the substitution of an 80386 CPU result in correspondingly faster execution speeds. The PC Intel 80286 or 80386 CPU is used as the display and storage controller and system supervisor. Display on a compatible monitor uses the EGA graphics standard with 640x350 pixel resolution. A gray scale can be used with monochrome EGA, such as that found on Supertwist LCD and gas plasma screens common to portable computers, but a color display, which maps signal amplitude to a color palette, makes interpretation easier for the user.

The VOICE spectrogram uses the SKY321-PC fixed-point (integer) digital signal processor, which includes the Texas Instruments TMS32010 digital signal processing chip as the numerical engine. As the data memory is double ported, it can be accessed simultaneously by the PC and the TMS320. An efficient memory controller can accomplish simultaneous accesses with only occasional wait states issued to the PC or the TMS320. This efficient memory management scheme was the primary motivation for selecting the SKY321-PC add-in board. Since relatively large amounts of data are moved across the interface at each frame, the data handling efficiency becomes more important than the relative clock speed trade-offs between various TMS320 chip versions.

The analog data is acquired and digitized by the MetraByte DASH-16 A/D board; however, any PC-compatible board with a DMA capability can be accommodated. At acquisition rates of up to 100kHz, the DMA overhead becomes significant, adding up to two seconds to screen scroll time for an 8-bit DMA board. For this reason, PC-AT compatible boards with 16-bit transfer capability are preferred.

Optimizing the Hardware Interfaces

For a spectrogram to be displayed in real time along with an audio signal, a fast scrolling and processing rate is critical; details of the spectrogram would be lost without this turnaround speed. To achieve this during real-time mode, there are three independent subsystems operating: the DMA activity, the TMS320 numerical data processing, and the CPU data transfer and screen display routines. Care was taken to balance the load between the TMS and the Intel 80286 processors; the efficient coupling of these two processors largely determines the ultimate speed of the real-time display and the usefulness of the spectrogram. The 80286 is concerned primarily with data formatting and display, the TMS with the numerical routines.

To reduce acquisition overhead, the A/D board is used in a continuous DMA mode, where a circular buffer is repeatedly loaded with newly acquired data. Up to 384k (six 64k pages) of memory storage is made available to the acquisition DMA, and it is continuously

filled with the data. The DMA is redirected to a new page after a terminal count (TC) is received upon completion of a page. The TC is wired to a system interrupt; the interrupt routine assigns a new DMA page to the data acquisition buffer, and updates the buffer list. Only one instruction is required to reassign a page: only the page register needs to be modified. The principal advantage of this scheme is the availability of a relatively large buffer, with no requirements for external memory cards. The CPU polls the DMA IC for the most recently written address and downloads the most recently written data segment to the TMS. When expand mode is entered, the DMA activity is stopped, and the memory buffer referenced from the last address written. This method is relatively simple to implement, and guarantees that there will be no lost samples, since the DMA is writing continuously. This allows for work on complex data sets, such as high bandwidth vocalizations (e.g. dolphin whistles) or long records such as whale songs.

The PC transfers the data between the circular data buffer and the TMS320. The acquired data typically is offset binary, which is converted to two's complement and normalized with a software-selectable gain before being downloaded to the TMS.

The TMS320 operates on a data buffer downloaded to its memory and outputs a flag signifying that a completed data buffer has been placed at a predetermined location in its memory. The location of the input and output buffers is controlled by the PC microprocessor, which implements a double buffering scheme so that a set of output and input buffers is manipulated by the PC while the TMS320 is operating on a distinct set. The buffers are flip-flopped at each frame.

Program Structure

The software was developed within the MS-DOS 3.2 operating system using Microsoft Version 5.0 C compiler and Microsoft Macro Assembler Version 4.0. The Version 5.0 C compiler was used specifically to take advantage of the graphics functions that were introduced at this version level. In addition, routines unique to the SKY signal processing board are integrated into the VOICE software structure. The SKY321 environment includes a host-resident SKY321 macro operating system, a SKY321 macro preprocessor, and a SKY321 assembler.

The primary functions of the main program are to control interaction with the user via the initial command line and the keyboard; output processed data to the screen and, optionally, to save raw data to a disk file; and to accept and transfer data addresses. In the initialization section, the program also loads program and data files to the TMS board and turns on the digitizing card.

The CPU controls all output to the monitor. The EGA driver is accessed directly with register commands; a fully register-compatible graphics subsystem is required. To address the hardware through DOS interrupts would reduce the display speed at least threefold, rendering the system useless for practical bioacoustic analysis. Significant effort was spent to insure careful optimization of the graphics routines.

The dimensions of the screen affect the display scrolling rate, so they were chosen to facilitate screen speed, and to allow sufficient space for annotations on the margins of the screen. The EGA map allows for a screen display of two windows, each with 128 pixel rows,

one above the other, along with system configuration information. The writing of pixels directly to the EGA video RAM is a two-step operation. When each 256-point array has been returned to main memory from the TMS board, with each element of the array coded for color, half the array (128 points) is sent as a column of pixels to the righthand end of the display area, where there is space available for eight pixel columns. When this group of columns is filled, the entire display is moved left by eight pixel columns, so that the lefthand array set scrolls off the screen, and room is provided on the right to receive new pixel columns of data. The factor 8 was chosen because EGA treats 8 horizontal pixels as a unit; each EGA unit is 8 pixels wide by 1 deep. This characteristic enables smooth scrolling from top to bottom of a display; however, the application for which VOICE was developed required scrolling from right to left, the usual convention with bioacoustic/speech analysis display tools.

The scrolling routine required optimization, as each data point representing a screen pixel must be physically remapped to a new location at each screen scroll. As the screen windows are 128 x 480 bits each, and a screen scroll time represents the time for a single pixel to move across all 480 columns, the screen scroll routine could be a system bottleneck. Fortunately the EGA standard provides for a 32-bit (8-pixel) move with a single (block move) instruction, and this allows the screen to be scrolled in three seconds when there is no other system activity on an EGA system with no wait states at 10 MHz operation.

The screen write commands are not as critical as the scrolling. The bit-mapped data are written to the screen with direct register command, and the stationary information outside the windows is handled through the high level graphics library provided with the Microsoft Version 5.0 C compiler. These screen annotation functions are used to display cursors, draw and fill colorcoded boxes, and to write prompts and data values to the screen margins surrounding the spectrogram display.

Interaction between the PC and the SKY321 occurs on two levels, through programs executed on the PC, and through those executed in the SKY board. The full list of SKY modules used by the VOICE program is listed in Table 1. The execution of the FFT processing programs is assigned exclusively to the SKY board. The only role played by main program VOICE is to download the appropriate program and a sine table for all transforms up to 1k, during execution. This is done just once, at initialization. A library of FFT programs is available, reflecting such parameters as the display complexity and color maps, and the data buffer/FFT size. The PC microprocessor selects the appropriate program based on the configuration parameters. The TMS320 programs, written in TMS320 assembly language, are carefully optimized, as their execution can be a significant bottleneck to processing speed. The SKY library function modified for VOICE is FT256, a complex Fast Fourier Transform. In order to optimize the division of processing resources between the PC and the SKY board, FT256 was augmented to compute squared magnitude of the FFT values and to assign color codes to the processed data. The values returned to the PC from the SKY board could immediately be sent to the EGA display without further processing. The FFT library function was adapted for the spectrogram display. The program remains active in the SKY board throughout a session, repeatedly processing data sent to it from the PC, and returning the output values on command from the main VOICE program.

Table 1
Integration of SKY321 Modules with Program VOICE

	Description	Implementation
fftcolor.320	FFT program executed on SKY board	downloaded to SKY board during VOICE execution
sintab.dat	1k FFT sine table used by SKY board for FFT calculations	downloaded to SKY board during VOICE execution
hsmos.h	group of functions to control interface (I/O) between PC and SKY board	included in main program VOICE as header file; invoked by VOICE program
S32asm.obj	linkable SKY object module	included in VOICE link command to enable hsmos routines

SKY supplies routines for initializing and controlling the SKY321, the "Host-resident SKY320 Macro Operating System" (hsmos). The main VOICE program controls execution of these modules. The file hsmos.h includes 14 functions which start and stop TMS processing, and transfer data between the TMS and the PC. The reference manual gives details about eight of these functions; we found this scheme to be restrictive, and so bypassed several of the eight top-level functions to directly address the lower level functions. The source code is commented sufficiently to make this course of action reasonable. While the processing program is running, the TMS is stopped briefly by the PC once per data frame in order to update the input/output buffer locations for the next frame. These parameters must be loaded into the TMS program memory, and the SKY PC board requires that the TMS be stopped before its program memory can be accessed by the PC.

Operation of the digitizing card is controlled by a group of C functions from the main program; they turn the digitizer on or off, and read the last address accessed by the CPU's DMA. Input data are sent directly from the DMA board to the SKY board, with the main program VOICE specifying the addresses of the input buffer and the TMS buffer. The digitizing card and the TMS board thus are coordinated so that data values are transferred to the main program only once — when they are ready for display.

The source code for VOICE is comprised of the main program and multiple functions written in both C and assembly language; a brief description of each function is given at the end of this report. The SKY hsmos.h header file is included in the main program so that it can be compiled by the C preprocessor. Each module is compiled using Microsoft C large model. We collected all object files in a library, vce.lib. Note that the FFT program fftcolor.320 is not part of the vce library; it is a binary file which was assembled using the SKY assembler SKYMPP during program development. The compiled VOICE package is linked with SKY320 object code (S32asm.obj), supplied by SKY. The SKY system requires that the /ST32767 option be specified during linking. As an example, here is some sample code to compile a C function named labelv.c, add it to the library, and then link it with the SKY routines to produce an executable version of VOICE:

```
CL /AL /c labelv.c
lib vce-+labelv;
link /ST32767 voice+S32asm,voice,vce.lib;
```

Table 2
Installation Guide
MetraByte DASH-16 high speed A/D converter board

Before installing the board in the machine, set the following switches, using a small screwdriver or a pen. Do not use a pencil.

Base Address switches:
1 - off
2 - off
3 - on
4 - on
5 - on
6 - off

DMA slide switch: level 1
CHAN CNFG (channel configuration) slide switch: 8
A/D slide switch (controls input range): BIP (bipolar)

Gain: Set for appropriate input range

switch	+/-10v	+/-5v	+/-2.5v	+/-1v	+/-0.5v
1 -	on	off	off	off	off
2 -	off	on	off	off	off
3 -	off	off	on	off	off
4 -	off	off	off	on	off
5 -	off	off	off	off	on

SKY321-PC Coprocessor Board

Set data memory base address to D000: at JP2, jumper 3 - 4.
Set program memory base address to C800 : at JP4, jumper 3-12, 4-11, 6-9, 7-8.
Install the A/D board and the SKY coprocessor board in any free full-width slots.
Copy all SKY software into a \TMS directory. Copy dsp321.dat from the VOICE floppy to the \TMS directory.
Test the coprocessor board by running TST321. Be sure the test runs at least twice.

Using the Spectrogram Instrument

Installation

Installation of the hardware should be done by someone who is able to remove the cover from the PC and install the two accessory boards. The MetraByte DASH-16 A/D board has switches and jumpers that must be set before the board is installed in a slot. Instructions for setting these are given in Table 2. The A/D system also includes a box external to the PC, a "Screw Terminal Accessory Board Model STA16". The input cable from a tape deck or microphone feeds into this box; a ribbon cable from the box then is plugged into a connector on the edge of the A/D board. This configuration may be modified so that the input cable runs through an antialiasing filter before reaching the STA16 box. The audio signal can be heard while the spectrograms are displayed if a split input cable is used, with one cable attached to the STA16 box and the other to an amplifier. The SKY321 board must have two base addresses set; it is then installed simply by sliding it into any available slot in the PC.

In order to run the program, the following files are required in the current directory:

voice.exe	main executable program
fftcolor.320	TMS320 executable program
sintab.dat	TMS320 FFT sine table

Running the Program

The program may be run simply by typing "voice" from a directory containing all the above files. An audio line output standard signal (1.0v RMS = 0 dB) at the digitizer input is assumed. The sample frequency is 50kHz, resulting in a signal display of ten colors from DC to 25kHz, with the top window displaying the spectrogram of the signal, and the bottom window showing a narrow waveform envelope, with the clipping window activated. Default values delimiting each color are 1, 2, 4, 8, 16, 32, 64, 128, 192, 256, 320, and 32767, which display the optimum spectrograms of ocean mammal sounds. No data are saved. The program is exited by typing 'x'.

Interaction with the VOICE program is on two levels: through command line options, and through specified keystrokes while the display is running. The command-line options configure the system with desired operation details, i.e., optimized parameter settings for particular animal vocalizations or data acquisition modes. Once running, the program is interactive through the keyboard.

Table 3
Command Line Options

-cc	change color map via the colors.dat file
-cl	change color threshold level via the levels.dat file
-f1	sample one channel; display spectrogram in top window
-f1b	sample one channel; display spectrogram and a bar graph indicating RMS signal amplitude (default)
-f1e	sample one channel; display spectrogram and the lower window of the RMS signal amplitude
-i #	set the size of the incremental step used to read a file of spectrogram data
-n #	set number of points per transform, zero pad the rest (default = 128)
-r	assign a file of previously saved spectrogram data to the input stream
-s #	set sample frequency in kHz, using an even number; displayed spectrogram is half the sample frequency (default = 50kHz)
-t #	set interval (seconds) for display across top of screen (default = 1.0)

Command Line Options

The command line options can be invoked on line at run time, or listed in a batch file for repeated use. If the system is to be used in one mode only, and ease of use is a prime factor, the defaults can be changed (in function cmdopt.c) to reflect the required features, and the software package recompiled and linked. A typical command line might be:

VOICE -f1e -t .5 -s 15

This particular line would result in a display of a spectrogram across the top of the screen, with the waveform envelope in the lower window stretching from one border of the window to the other; time ticks across the top of the screen every half second; and a sampling frequency rate of 15kHz (instead of the default, 50kHz).

The display of a spectrogram in the top window, with the signal envelope appearing in a bar-type display at the right of the bottom window, is the default, specified in the list of command line options as "-f1b". This mode is particularly useful for fast scrolling of a single spectrogram. When the display leaves real-time mode, and is used to examine data stored in memory, the bar indicator is replaced with a signal envelope the full width of the lower window; during memory operations, speed of scrolling is not a factor. The real-time spectrogram display can also be used with a signal amplitude display which stretches the full width of the bottom window ("-f1e") when the signal waveform is of inherent interest, such as for voice amplitude analysis, or when it is critical to guard against clipping of the digitized signal.

The sampling command "-s #", where # represents some number, determines the digitizer sampling frequency. In order to determine the proper sampling frequency for a particular signal, you must first determine the maximum frequency at which the signal contains energy. This can be done using VOICE by setting the sampling rate higher than twice the likely highest frequency of the signal, and then using the spectrogram to measure the highest frequency. The sample rate should be set to 2.5 times the highest frequency, or higher. This is a critical factor in achieving a meaningful spectrogram. For example, a dolphin whistle with a maximum frequency of about 20kHz produces a clearly defined spectrogram at a sampling rate of 50kHz. At the same sampling rate a humpback whale song is scarcely visible; however, if the sampling frequency is set at 2 or 3kHz, the same whale song appears in full detail. The spectrogram display on screen is half the sampling frequency; a 50kHz sampling rate yields a 25kHz spectrogram signal. The desired sample frequency is entered in kHz. Note that the system may not be able to synthesize the exact frequency requested for all cases, since the sample frequency is derived by integer division of a 1 MHz oscillator. In that case, the closest available frequency is selected. With the the DASH-16 board, the maximum sample frequency is 100k samples/sec. If the desired sample frequency exceeds the board capabilities, the highest possible sample frequency is selected. Once the sampling rate has been chosen, either by taking the 50kHz default or by using the "-s" option, that rate remains in place for the entire session. To change the rate, the user must exit from the program, and start VOICE again.

For very low frequency sounds, the display can also be sharpened by using the "-n #" command. This selects the number of input points for each FFT; the default is 128 (256/2).

The FFT size presently is fixed at 256; the "-n" command establishes the number of points within the 256-point transform. If n points are used, the rest of the FFT input is zero-padded. Generally, decreasing the number of points sharpens up the display of broadband transients at the expense of overall display quality. This effect is particularly noticeable with very low frequency impulsive sounds, such as fish grunts.

Time ticks across the top edge of the top window mark the elapsed time for sections of the display to scroll across the screen. The default is for one second between ticks; with the "-t #" command it can be changed as desired. Choices should be entered as decimals, e.g., .5 for half a second.

Data saved to file during a previous session — an operation described in the section on interactive keystrokes — may be displayed by running VOICE with the "-r" command line option. The exact format is

voice -r somefileid

A default for spacing through the disk file has been set which provides a display to fill the entire window width when the source is a disk file of "moderate" size, an arbitrary choice by the programmer. If the user is faced with a vertical sliver of color when he attempts to review some saved file, the saved data file is probably far smaller than the "moderate" size. Such a file can be viewed by using the "-i #" option. The choice of number to replace the pound sign is an estimate which the user will learn to make with experience. Since the default "i" number is 138 x 6 (828), a good place to start is 138.

The VOICE package also includes two data file templates which can be used to change the color spectrum in the spectrogram display — colors.dat — and to alter the levels at which colors change — levels.dat. They may be modified using any editor to suit the user's requirements. Note that the choice of levels should be keyed to the output of the FFT processing, which produces maximum values lower than those of the raw data. Restrictions on these data files are:

- a maximum of 16 values may be used in each file
- the number of colors and levels used should be the same
- color values following the last used must be "63"
- the level value following the last used must be "0"

If these templates are to be used, the program needs to be informed by the use of the command line instruction "-cc" for a color spectrum change, and "-cl" for a change in the color threshold levels.

Interactive Commands

The interactive commands are entered on the keyboard while the program is running. They were designed specifically to make the program easy and natural to use. All interactive messages and prompts appear in a window at the lower right of the screen; if any illegal commands are entered, a message is displayed, listing the keystrokes that can be used at that juncture. On-screen explanations of the keystroke functions can be displayed in a help window which appears when the user strikes the 'h' key. The keystroke list varies depending on which mode of the program currently is operative; only those commands directly applicable are displayed. Table 4 summarizes the commands, and flags each command by mode. If the command is relevant to a running display, while data streams across the display window, it is flagged as "realtime." This running display can be stopped at any time for closer examination of a spectrogram in "memory" mode. Commands tagged as "global" are valid in both modes. During real-time display of data, the relevant commands are 'x', 'f', 'h', 'm', and <del>. This is the initial default mode when the program is started. If the signal which appears on screen merits closer examination, touching the 'm' (memory mode) key will halt the flow of new data and invoke a display of up to 384k of data stored in the program's memory buffer — the same data that was on screen when the 'm' key was hit. To return to the running display, the user touches the spacebar.

The memory-mode commands are all related to functions which operate on the 384k of data which were captured in memory when the 'm' key was hit. This buffer full of data can be recalled to the screen for careful examination, expanded for a study of details, and saved to a disk file. The commands enabled during memory mode are the cursor keys, 'h', 's', 'x', <esc>, <del>, <enter>, <space>, and the signal gain controls.

Hitting the 'x' key on the keyboard terminates the program and returns to DOS. Improper program exit, such as the use of 'Ctrl C', may leave the data acquisition DMA running, with disastrous consequences to subsequent operations. If VOICE crashes the system upon exit, it is probably because the DMA activity was not stopped during a nonstandard exit.

The 'f' key will freeze the spectrogram display. This feature is useful if a scrolling spectrogram deserves further scrutiny, or is to be plotted by dumping to a dot matrix printer. The two requirements for making such a plot are that the printer allows graphics mode, and that the DOS command "graphics" or "crtdump" previously has been invoked (usually in the autoexec.bat file). The user should be aware that data will continue to stream through the memory buffer while the spectrogram display is static on the screen; if the user's next action is to display the contents of memory, they may be very different from the screen display at the time the 'f' key was hit. After a freeze screen, the usual action is to hit the spacebar and return to a real-time display.

Both the save and the expand capabilities of the program depend on an initial display of the memory contents, which occurs when the user hits the 'm' key. Data acquisition is halted by this action so that the contents of the memory buffer will be available for a series of displays, and for saving to a file. The display is calculated so that the spectrogram derived from data in the memory buffer always fills the screen, no matter what percentage of the buffer has been filled with data. All the new data in memory scrolls across the screen, stopping when it reaches the left margin. If the entire 384k buffer has been filled, the

Table 4
Interactive Commands

mode	key	function
global	x	exit the program
global	h	display help window
realtime	f	freeze the screen (static display)
realtime	m	display spectrogram of data currently in memory
memory	s	save data delimited by cursors
memory	< *F1* >	move left cursor to the left
memory	< *F2* >	move left cursor to the right
memory	< *F3* >	move right cursor to the left
memory	< *F4* >	move right cursor to the right
memory	< *enter* >	signal that cursor positions are final
memory	< *esc* >	recall the previous screen
global	< *space* >	restart the real time display
global	< *del* >	erase clipping light below the signal amplitude display
global	↑	increase signal gain by 3 dB
global	↓	decrease signal gain by 3 dB
realtime	→	decrease scrolling speed
realtime	←	increase scrolling speed

spectrogram is somewhat compressed in order to fit on the screen in its entirety; however, if the user should hit the 'm' key before the buffer has been completely filled, only new data will be used, so that the spectrogram may be expanded as it stretches from one side of the window to the other. A partial buffer display can occur when a user requests a new memory display immediately after leaving an earlier display. At a low sampling rate, such as 10kHz, it takes a long time for the buffer to refill.

When the screen has filled with the memory display, line cursors appear at each edge of the window. Thereafter several options are available until the screen is returned to a real-time display: expansion displays of portions of the memory buffer data; saving to disk file of any portions of that data; redisplay of earlier screens of spectrograms — the "recall" feature.

The cursors allow the user to take advantage of the program's capabilities for enlargement. The <F1> and <F2> keys are assigned to the left cursor, the <F3> and <F4> keys to the right cursor. Each tap of an F key moves the line cursor eight pixel columns (the width of a text column). Holding down a key results in a quick succession of moves by the cursor. When both cursors delimit the portion of the spectrogram that is of interest, the user may choose to save that portion by hitting the 's' key, or to hit the <enter> key in order to see that portion expanded to fill the window. This expand capability can be used on each screen display as the data are enlarged repeatedly. This allows for the examination of events too detailed to be observed in real time. For instance, a whale click 1/10th of a second long can be located on the real-time display, but the details of the amplitude and frequency distribution can be seen only when the spectrogram has been expanded.

Each expanded display fills the screen window completely, from side to side. The entire display is bracketed by vertical bar cursors which delimit the start and end of the displayed data. The exact matching of the data display width to the window dimensions enables the system to track changing start and end positions in the data as the cursors are moved. The requested data are sampled in 480 evenly spaced data segments, using a calculated step to advance the starting location of each segment; the screen window is 480 pixel columns wide. This offset is calculated by dividing 480 into the data length — the number of points between the start and end of the delimited data. The resulting step value must be an even number so that the data values can be read in pairs of sine and cosine. If the value is an odd number, it is decremented to the next lower multiple of 4. In theory, the step between starting values could be a minimum of 4. In practice, the scheme produces exactly 480 columns of pixel data for interval steps of 20 or greater (at least 9600 bytes of data, or 2400 groups of sine-cosine pairs). When steps are smaller than 20, the need to decrement a step value combines with granularity problems to prevent division of the available data into 480 even sets. The result is an excess of data columns — too much data. In the example below, the user has requested 9456 data points.

9456. / 480. = 19.7
Sample period must be 16

With a sample period of 16, the 480 pixel columns would display 7680 points; there are 1776 points left over. One choice would be to lop off the extra data to force an exact fit on the screen. However, in the interest of veracity we chose to exclude displays which would overflow past the left margin. When a display cannot be made, a message appears at the

lower right of the screen; although the data cannot be displayed, it can be saved to a disk file. If the save is not wanted, the previous spectrogram is sent to the display window.

The 's' command saves to file the section of data delimited by the cursors. The data saved is the raw offset binary digitized data. A prompt requesting a filename appears at the right; the user enters his choice of filename, followed by the <enter> key. If the desired file already exists, the new buffer is appended to it. This mode is useful, for example, in cases where the digitized data is sorted by species, such as when scanning a tape containing dolphin and whale calls. Examples of each can be separated into the respective files and saved. The disk writing operation begins immediately, as indicated by the message on screen. When it is completed, the message announces this fact, and the user is then free to move the cursors for another expansion (if the current display segment is not too small), make another save, return to the real-time display, or to recall an earlier display.

The "recall" feature is the reverse of the expand operation. After several screen expansions, the user may wish to return to a screen display which occurred early in an expand series. The display which immediately preceded the current display can be recreated in the window by hitting the <esc> key. This key can be used repeatedly to step backwards through the expand series until the original display of memory is reached.

The arrows on the cursor pad may be used to increase/decrease the signal gain in software — the up and the down arrows — and to increase/decrease scrolling speed — the left and right arrows. These arrows will work only when the <NumLock> key is off. The gain is a software value, initially set to 0, which is incremented or decremented by one for each keystroke; the value can be either positive or negative. The input data values are multiplied by 2 raised to the power of the gain value before they are processed by the FFT operation. Note that scrolling speed cannot be increased to more than the default; the speed increase key is useful only when the scroll speed previously has been decreased. The gain and the scrolling speed features can be invoked only during a real-time display; they are not enabled for a display of memory.

Often it is critical to know whether the digitized input signal is clipping. This can occur when the gain of the analog signal saturates the A/D converter. A clipping indicator has been included as a monitor for this condition. It is enabled in the default mode of VOICE where the signal amplitude is displayed along with a spectrogram. If clipping occurs, the waveform appears to overflow into a small vertical bar on the lower right of the screen. When this warning of too much gain has appeared, it can be erased by hitting the <del> key.

Future Development and Applications

The VOICE program, which initially was developed to answer a specific need, has evolved into a versatile tool for a growing number of applications. Data may be input from analog tapes, from a live signal via a microphone, or from disk files holding binary data. All sources produce spectrograms, the main function of the program. Data may be viewed in a variety of modes — streaming from right to left, halted on screen, or expanded. Depending on the application, the spectrograms may be viewed with no saved output; delimited so that specified raw data from the input source is saved to disk file; or the screen display may be reproduced on a printer. The screen display itself can be manipulated; the amount of information displayed is determined by the user's choice of whether to view the signal waveform; and the setting of the interval between the time ticks across the top of the screen permits estimation of the length of each signal.

At present the program can display one channel of data; the program already has the "hooks" to add the capabilities of a two-channel and a four-channel display. Sampling frequency is now limited to a maximum of 50kHz by the A/D board; since the rest of the system can handle up to 100kHz, this constraint could be removed by use of a different A/D input board and the replacement of the present data acquisition subroutine in the software package. Presently the largest section of data that can be saved with a single command is 384K (six 64K pages), but subsequent saves can append data to the same file; with the addition of extended memory to the PC and some changes in the code, a larger section of data could be saved in a single operation.

These are a few possibilities for expanding the capabilities of VOICE. In the short time that it has been available to WHOI investigators, we have made a number of adaptations, some as simple as changing the defaults. We encourage potential users of the system to use VOICE in its present form, or to adapt it to different PCs or boards. Listings of all sources, the executable VOICE program and required files are available on floppy disk upon request.

References

SKY321-PC & 320-PC (rev.4) Reference Manual. 1987. Document #321-PC-RM-87-1.2 SKY Computers, Inc., Lowell, MA 01852.

DASH-16/16F Manual. 1986. MetraByte Corporation, Taunton, MA 02780.

Disk Operating System Technical Reference, Version 2.10. 1983. IBM Personal Computer Language Series. Microsoft Corp.

Kliewer, B. D. 1988. EGA/VGA, A Programmer's Reference Guide. Intertext/McGraw-Hill, New York.

Source Code Listings

Name	File	Description
main	voice.c	Main program
voice.h	voice.h	Header file for main program voice.c
keydefs.h	keydefs.h	Header file for modules getkey, getmem, keys, lcursor, review
blkbox	labelv.c	Draw a black and white box on screen
boxes	labelv.c	Draw several columns of color boxes
calcstep	calcstep.c	Find step to use in memory buffer read
chanenv	chanls.c	Display spectrogram and signal amplitude
chcolor	change.asm	Establish color palette
clearhlp	helpvce.c	Erase "help" window and contents
clearmsg	messages.c	Erase text from message center box
cmdopt	cmdopt.c	Handle command line options
dashget	dashin.c	Get offset for current data
dashin	dashin.c	Start data acquisition via DMA
dashoff	dashin.c	Stop DMA data acquisition
delmag	delmag.c	Delete amplitude clipping light
endint5	endint5.asm	End DMA end-of-page interrupt condition
erase	erase.asm	Erase contents of lower window
getkey	keys.c	Identify key hit by user
getmem	getmem.c	Set up for display of memory buffer data
handle	handle.asm	Handler for DMA end-of-page interrupt
helpvce	helpvce.c	"Help" window text
kayhdr	kayhdr.c	Prefixes saved data with Kay Sona-Graph format 5500
keyopts	keys.c	Enable interactive key options
labelv	labelv.c	Draw color code boxes and annotations
lcursor	lcursor.c	Enable movement of cursors
linecurs	lcursor.c	Draw a vertical white line (cursor)
messages	messages.c	Text for message center box
movefull	move.asm	Move data column in both windows to left
movetop	move.asm	Move data column in top window to left
movtolft	bounds.c	Find data start and end addresses
movtorit	bounds.c	Find data start and end addresses
onechan	chanls.c	Display one frequency channel
putcurs	putcurs.asm	Draw a vertical line cursor to screen
putlcurs	putlcurs.asm	Draw a left-bracket cursor to screen
putrcurs	putrcurs.asm	Draw a right-bracket cursor to screen
putft	labelv.c	Write sampling frequency value to screen
putmag	putmag.asm	Draw signal amplitude display to screen
review	review.c	Read/display data file saved by VOICE
savscr	savscr.c	Save data to disk file
scrntop	screen.asm	Draw spectrogram to top window

setscr	screen.asm	Reset palette to default colors
showdat	showdat.c	Display spectrogram for expand modes
totms1	totms.asm	Move data from DMA buffer to TMS memory
txtprep	txtprep.asm	Configure screen to allow graphics text

```
/*   Copyright @ 1989 by A. Martin and J. Catipovic
     All rights reserved.

VOICE.C --
          A spectrogram software package designed for use with a PC-AT
personal computer (with hard disk) that is equipped with an EGA graphics
board, a Sky321-PC signal processor board, and a MetraByte DASH-16
A/D board.

       For a full description of the spectrogram instrument, a user's
guide, and a description of this software package, see:
       VOICE -- A Spectrogram Computer Display Package, by A. Martin,
J. Catipovic, and P.L. Tyack, 1989. WHOI Technical Report WHOI-90-22.

                        ****************

This version uses the TMS HSMOS.
HSMOS requirements:
    sintab.dat and fftcolor.320 must be available at runtime.
    Compile as a large model, with hsmos.h and hsmos.def
    Link with S32ASM.OBJ, and with /stack:32767

 NB: To change the TMS board address for data memory, change:
        - hsmos.def
        - tmsaddr variable in voice.c

VOICE compile requirements:
       voice.h
       vce.lib - all object code for this package
*/
/**********************************************************************/

#include "voice.h"

main(argc,argv)
 char *argv[];
{
 short i,j,k,n,m;
 int option, dtyp, tmark, numchan, dmapgtmp, backoff;
 unsigned int clrbits, dashtmp = 0;
 float fmax, ftmp;
 time_t start,finish;

 absptr = (int *)absaddr;
 tknt = cycle = i =  n = m = 0;
 envelope = j = 0;
 speed = gain = 0;
 lp1 = locparm1;
 lp2 = locparm2;
 dtyp = 0;
 tmark = 0;
 scrntime = 0.0;

                  /*INITIALIZING*/

/*pick up the command line options, if any, plus some global
  initializations*/
    if(( option = cmdopt(argc,argv)) < 5 )
       move = movefull;
    else
       move = movetop;

/*Set up the byte count to back up before sending data to tms board*/
```

```
    backoff = 512;
    clrbits = 0xfffc; /*create a number divisible by 4*/

if( option == 8)
    dtyp = 2; /*saved output is colorcode values, not data -- not enabled*/


/*calculate sampling frequency*/
    fmax = (1000./((double)(knum1 * knum2)))/2.0;

/*clear TMS data memory*/
      tmsptr = (unsigned *)tmsaddr;
      enable_p0;
      for (i=0; i<32000; i++)
          tmsptr[i] = 0;

 /*load in fft program, sinetable and color level table*/
      if(option == 2)                  /* 2 sample channels - not enabled */
        {
        /*lfile("ftcolor2.320",0,1500,PMEM); */
        numchan = 0x10;
        }
      else if(option == 1)             /* 4 sample channels - not enabled */
        {
        /*lfile("ftcolor2.320",0,1500,PMEM); */
        numchan = 0x10;
        }
      else
        {
        lfile("fftcolor.320",0,1500,PMEM); /*1 sample channel*/
        numchan = 0x00;
        }

      lfile("sintab.dat",locparm1[2],locparm1[2]+2048,DMEM);
      writdm(locparm1[5],edge,32);

/*write labels and annotations to the screen*/
      labelv(option);

/*write sampling frequency to the screen*/
      putft(fmax,scrntime,option);

/*set up for timing ticks on screen display*/
  now = (double)((newsec=clock())/ctick);

                        /*EXECUTING*/

  if(option < 10 )
     dashin(absptr,numchan);  /*  get analog data*/
  else
     review(option); /*display stored binary file, and exit*/

  dashtmp = dashget();

/**********************Top of Read-Execute Loop***********************/

/*set up for keyboard interrupts*/
   while(1){
      if(kbhit() )
          keyopts(numchan,option,dtyp);

/*load starting addresses of input data, output, sintable, color lookup
  into 30 - 35 of Program memory*/
      writpm(30,lp1,12);
```

```
/*Use an HSMOS routine to start processing*/
      strt320(30);

/* get the address of the input array so that the TMS can find it*/
      dashtmp = dashget();
      dmapgtmp = dmapage;

/* 'backoff' must be subtracted from the current offset, so the resulting
   pointer may be on the page before dmapage*/
      while( (dashtmp & clrbits) < backoff )
              {
               dashtmp = dashget();
               }
      dashtmp = (dashtmp - backoff) & clrbits;

      absaddr = ((long)dmapgtmp)<<28;   /* this defines the segment*/
      absptr = (unsigned *)(dashtmp | absaddr);/*dashtmp is the offset*/

 /*here's the pointer for the TMS board*/
      tmsptr = (unsigned *)((lp2[0]<<1) | tmsaddr);

/*download the data array to memory in the TMS board*/
      enable_p0;
      tmsfmt(absptr,tmsptr,ptknt, gain);

/*display timing ticks across top of screen*/
      if((double)( (newsec=clock())/ctick) != now)
        {
         now = (double)(newsec/ctick);
         tmark = 1;  /*make a time tick on the screen*/
         }
      else
         tmark = 0;

/*display channels of frequencies*/
      channel(tmark);
      if( cycle >= 8)
      {
        move();
        cycle = 0;
       }

      for (i=0; i<speed; i++);  /*controlled by <-,-> keys */

/*Read the results from the TMS320 into output array fftval*/
      readdm(lp2[1],fftval,fftout);

/*swap the location parameters*/
      ltmp = lp2;
      lp2  = lp1;
      lp1  = ltmp;

/*wait for TMS320 to finish - bit 3 is set in STCREG(IOBASE)  */
      while((inp(IOBASE) & 0x08) == 0);
      hlt320(); /* TMS board is all done for this pass*/

     }           /*end of read and execute loop*/

}
/*********************************************************************/
/*********************************************************************/
/*VOICE.H
  An include file for voice.c.
```

```
*/
/**********************************************************************/
/* include files*/

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <conio.h>
#include <math.h>
#include <time.h>
#include "hsmos.h"
#include "keydefs.h"

/**********************************************************************/
/*defines*/
#define MAXSIZE 512
#define BUFSZ 20

/**********************************************************************/
/*declare functions*/

 unsigned int chanenv();
 unsigned int onechan();
 unsigned int twochan();
 unsigned int scrntop();
 unsigned int scrnbot();
 unsigned int movetop();
 unsigned int movefull();
 unsigned int handle();
 unsigned int totms1();
 unsigned int totms2();
 unsigned int review();
 unsigned int (*move)(), (*tmsfmt)(), (*channel)();
 unsigned int dashget();
 int dashin();

/*********************************************************************/
/*declare and define some global variables*/
/*for initializations dependent on command line options, see cmdopt.c*/

int *lp1, *lp2, *ltmp;
int ftsize,fftout,ptknt,inkey,tknt,lcol,rcol;
int dmapage, dmaknt, lineknt, doffinc, knum1, knum2;
int fftval[MAXSIZE],sintab[2048];
int gain, speed, cycle, kay, rknt = 0;
unsigned int *dataptr, *oldptr, *absptr, *tmsptr;
unsigned int envelope, doffset, endoff;
unsigned long int newsec, showaddr;
float tic;          /*user's choice of time mark spacing*/
double scrntime, now;
clock_t clock(void);
clock_t ctick;
FILE *stream;

char fidid[16];      /*filename to which data are saved*/
char newfid[15];
char fidin[BUFSZ];   /*binary input data (instead of analog)*/
char color1[256],color2[256],color3[256],color4[256];
char curcolor;

int locparm1[6] = {2048,6144,12000,0,0,30000};
int locparm2[6] = {4096,8192,12000,0,0,30000};
unsigned long int absaddr = 0x70000000, tmsaddr = 0xD0000000;
int edge[16]   = {0x0001, 0x0002, 0x0004, 0x0008, 0x0010,
```

```
                    0x0020, 0x0040, 0x0080, 0x00c0, 0x0100,
                    0x0140, 0x7fff,
                  0,      0,      0,      0};
int colors[16] = {0, 8, 33, 1, 9, 43, 15, 47, 61, 45,
                     37, 0, 63, 63, 63, 63};

/******************************************************************/
/********************************************************************/
/*KEYDEFS.H*/

#define XF  0x100

#define K_UP    72 | XF
#define K_DOWN  80 | XF
#define K_LEFT  75 | XF
#define K_RIGHT 77 | XF
#define K_PGUP  73 | XF

#define K_F1    59 | XF
#define K_F2    60 | XF
#define K_F3    61 | XF
#define K_F4    62 | XF

#define K_DEL   83 | XF
#define K_ESC    27
#define K_RETURN 13
#define K_SPACE  32

/*the following are values for lowercase letters*/
#define K_F     102
#define K_H     104
#define K_X     120
#define K_S     115
#define K_M     109


/******************************************************************/
/******************************************************************/
/* BOUNDS.C
   bounds.c -- These functions define the start and end locations of a
   portion of data ( in the memory buffer) that has been delimited with
   line cursors by the user.

Variables to be defined for getmem.c :
     again  -  kount of pages to be read (NB: pages are 4,5,6,7,8,9
               with beginning page containing data for the last page,
               so that 'again' for 6 pages = 6 for 7 iterations, 0-6)
     pagenow - starting page
     pagelast- ending page
     doffset - starting offset
     endoff  - last offset
*/

#include <stdio.h>

movtorit(tmpoff,loopknt,jknt,iknt,page)
 unsigned int *tmpoff, jknt, *page;
 int loopknt, iknt;
{
  unsigned int oldi;
  int i, j;

    /*Note: we must multiply offset * lcol * 8 using nested loops;
      else iknt is too big to fit into an integer*/
```

```
        iknt *= 8; /* 8 pixels per column move */

    for( j = 0 ; j < jknt ; j++)
     {
       for( i = 0 ; i < iknt ; i++)
         {
         oldi = *tmpoff;
         *tmpoff = *tmpoff + 1;
         /*now must see if tmpoff has gone from 65536 to 0*/
         if( *tmpoff < oldi )
           {
            *tmpoff = 0;
            *page = *page + 1;
            if( *page > 9 ) *page = 4;
            --loopknt;
           }
         }
     }

     return(loopknt);
}

/**********************************************************************/

movtolft(tmpoff,loopknt,jknt,iknt,page)
 unsigned int *tmpoff, jknt, *page;
 int loopknt, iknt;
{
  int i, j;

      iknt *= 8;

      for( j = 0 ; j < jknt ; j++)
          {
           for( i = 0 ; i < iknt ; i++)
            {
             *tmpoff = *tmpoff - 1;
             if( *tmpoff == 0 )
               {
                --loopknt;
                *page = *page - 1;
                if( *page < 4 ) *page = 9;
                *tmpoff = 65535;
               }
             }
           }

     return(loopknt);
}


/********************************************************************/
/********************************************************************/
/*CALCSTEP.C
 calcstep.c -- calculates incremental step to be used when reading data
               from memory buffer for options 'save' and 'prior' so that
               the resulting display will fill the screen exactly.
*/
/**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
```

```
calcstep(kmt)
  int kmt; /* 'again' in getmem.c */
{
  extern unsigned int doffset, endoff;
  int pagetmp, pagekmt;
  long unsigned int tmpstep, itmp;

  pagetmp = kmt;

  if( pagetmp == 0 )
    {
     tmpstep = ((unsigned int)max(endoff,doffset) -
                (unsigned int)min(endoff,doffset));
     tmpstep = ( ((float)tmpstep)/480.);
     }
  else if( pagetmp > 0 )
    {
     itmp = 65536 - doffset;
     tmpstep = endoff;

 /* see if the two partial pages added together make up more than
    a whole page (64k)*/

     pagekmt = 0;

     tmpstep += itmp;
     if( tmpstep > 65536 )
       {
        pagekmt = 1;
        tmpstep -=  65536;
       }

     tmpstep = (( ((float)tmpstep)/480.) + 0.5 ) + (pagekmt*138);

     if( pagetmp > 1 )
       {
        tmpstep += (pagetmp-1) * 138; /* for dmakmt > 2 */
        }

     return(tmpstep);
  }
}
```

```
/**********************************************************************/
/**********************************************************************/
;CHANGE.ASM
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)
;
_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,  _BSS,  _DATA
```

```
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT

PUBLIC _chcolor

_chcolor PROC FAR
;         bl holds palette register number
;         bh holds color value to be used

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di

        mov     di, [bp+6]
        mov     ax, [bp+8]
        mov     ds, ax

        mov     dh,0
        mov     cx,16                   ;loop counter

cloop:
        mov     ah, 10h                 ;set up for BIOS call
        mov     al,0
        mov     bl, dh                  ;register to set
        mov     bh, ds:[di]             ;color value
        int     10h                     ;enter the interrupt
        inc     di
        inc     di
        inc     dh
        loop    cloop

        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_chcolor ENDP

_TEXT  ENDS

END


/*********************************************************************/
/*********************************************************************/
/*CHANLS.C
  chanls.c  -- a collection of routines which control the screen display
               of spectrogram data and signal amplitude waveforms */
/*********************************************************************/

unsigned int scrntop();
#include <stdlib.h>

/*********************************************************************/
                        /* default display */
/* chanenv.c -- displays one channel of frequencies plus signal amplitude.
  May also be invoked with command line options -f1e or -f1b. Width of
  waveform envelope is determined in main and implemented by move.c*/
/*********************************************************************/
```

```
chanenv(tick)
  int tick;
{
    extern int ptkmt;
    extern unsigned *absptr;
    extern int fftval[], cycle, gain;
    extern char color1[];
    extern unsigned int envelope;
    int i,j;
    envelope = i = j = 0;

/*find the signal amplitude*/
    for( i = 0 ; i < ptkmt ; i++)
      if(envelope < absptr[i])  envelope = absptr[i];

   envelope = (envelope - 0x7fff) >> 8;
/* envelope = (envelope - 0x7fff) >> (8-gain); or use this to include gain*/

    for( i = 0 ; i < 128 ; i++)
        color1[127 - i] = (char)fftval[i];

    if(tick)
      {
      for( i = 124 ; i < 128 ; i++)
         color1[127-i] = 63;
       }
    color1[127] = 63; /* horizontal bar dividing display screen */

    scrntop(color1,cycle);

    putmag(&envelope,cycle);
    cycle++;
}

/*****************************************************************************/

/*  onechan.c -- displays one channel of frequencies */

/**************************************************************************/

onechan(tick)
  int tick;
{
    extern unsigned *absptr;
    extern int fftval[], cycle;
    extern char color1[];
    int i,j;
    i = j = 0;

    for( i = 0 ; i < 128 ; i++)
        color1[127 - i] = (char)fftval[i];

    if(tick)
      {
      for( i = 124 ; i < 128 ; i++)
         color1[127-i] = 63;
       }

    color1[127] = 63; /* horizontal bar dividing display screen */

scrntop(color1,cycle);

    cycle++;
}
```

```
/**********************************************************************/

/* twochan.c -- displays two channels of frequencies (option 2, command
               line option -f2).*/

/**********************************************************************/

twochan(tick)
  int tick;
{
    extern int fftval[],cycle;
    extern char color1[],color2[];
    int i,j;
    j = 0;
    for( i = 0 ; i < 128 ; i++ )
         {
         color1[127-i] = (char)fftval[j++];
         color2[127-i] = (char)fftval[j++];
         }

    if(tick)
      {
      for( i = 124 ; i < 128 ; i++)
         color1[127-i] = 63;
       }

    color1[127] = 63; /* horizontal bar dividing display screen */

    scrntop(color1,cycle);

    scrnbot(color2,cycle);
    cycle++;

}

/**********************************************************************/
/**********************************************************************/
/*CMDOPT.C
 cmdopt.c - picks up arguments entered on the command line at runtime
           and implements the user choices. It also does initializing
           of global variables not initialized in voice.h

 Command line options:
1)     -f4  = display 4 channels of sample frequencies - not enabled
2)     -f2  = display 2 channels of sample frequencies - not enabled
3)     -f1  = display 1 channel of sample frequencies
4)     -f1e = 1 channel of frequencies plus full width of envelope
5)     -f1b = 1 channel of frequencies, envelope a small bar graph (default)
8)     -w   = writes files of saved color-coded fft values (processed data). - not enabled
       -k    = saved files of raw data are in "Kay" 5500 format (headers)
       -t #. = interval tics ( < || ==  > one sec) for display on screen
       -cc   = change colors via data file Color.dat
       -cl   = change contour levels via data file Levels.dat
       -s #  = set sampling frequency. Default is the maximum, 50kHz.
       -n #  = number of words to send to the TMS board
       -i #  = size of incremental step for display of 'saved' data
10)    -r<space><somefileid>= review 1-channel data
11)    -r2<space><somefileid>= review 2-channel data - not yet implemented
*/
/**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
/**********************************************************************/

cmdopt(argc,argv)
  char **argv;
{
 extern char fidin[];
 extern unsigned int onechan(), chanenv(), twochan(), fourchan(),(*channel)();
 extern unsigned int doffinc;/*display increment*/
 extern unsigned int totms1(), totms2(), (*tmsfmt)();
 extern int edge[], colors[];
 extern int knum1, knum2, ftsize,fftout, ptknt, kay;
 extern clock_t ctick;
 extern float tic;

 FILE *fp, *fopen();
 char *p1,*p2,kntpt[3], newinc[4];
 int n,m;
 int dat1,opt,khz,x,nx,ny,flag,i;

/* Default values*/
 opt = 5;
 channel = chanenv;
 doffinc = 138*6;
 knum1 = 04;
 knum2 = 05;
 ftsize = 256;
 fftout = ftsize;
 ptknt = ftsize/2;
 tmsfmt = totms1;
 ctick = CLK_TCK;
 tic = 1.0;
 kay = 0;

      while(--argc > 0)
      {
          argv++;
again:    switch(argv[0][0])
          {
              case '-':
                      argv[0]++;
                      goto again;
            /*case 'w':
                      opt = 5;
                      break;*/
              case 'k':
                      kay = 1;
                      break;
              case 'n':
                      p1 = kntpt;
                      if (argc >= 2)
                       {
                        argv++;
                        argc--;
                        p2 = argv[0];
                        while (*p1++ = *p2++);
                       }

                      ptknt = atoi(kntpt);
                      if (ptknt < 0)
                         ptknt = 0;
                      else if (ptknt > ftsize)
                         ptknt = ftsize;
                      break;
```

```
case 'i':
        p1 = newinc;
        if (argc >= 2)
         {
          argv++;
          argc--;
          p2 = argv[0];
          while (*p1++ = *p2++);
         }

        doffinc = atoi(newinc);
        break;
case 'r':
     /*   if(argv[0][1] == '1')
          {
          opt = 10;
          tmsfmt = totms1;
          fftout = ftsize;
          }
        else if(argv[0][1] == '2')
           {
           opt = 11;
           tmsfmt = totms2;
           fftout = ftsize * 2;
           }  */

        opt = 10;
        tmsfmt = totms1;
        fftout = ftsize;
        p1 = fidin;
        if (argc >= 2)
         {
          argv++;
          argc--;
          p2 = argv[0];
          while (*p1++ = *p2++);
         }
         break;
case 's':
       khz = atoi(argv[1]);
       /*check for a decimal -- not legal*/
       if( (atof(argv[1])) > khz )
         {
          printf("\n**Sampling frequency must be an integer.\n");
          printf("  Please try again.\n");
          exit(0);
          }
       /*trap for sampling value greater than 50*/
       if(( khz > 50 ) || ( khz == 0 ))
         {
          printf("\n**Sampling frequency must be a kilohertz value between 1 and 50.\n");
          printf("  Please try again.\n");
          exit(0);
          }

       flag = 0;
       while(flag == 0)
       {
       /*find number which divides into 1000 to give khz*/
       x = 1000/khz;
       nx = 0;
       ny = 0;
       /*now factor out the x */
       i = 0;
```

```
                for(i = 2 ; i < x ; i++)
                  {
                  if(!(x%i))
                     {
                     nx = x/i;
                     ny = x/nx;
                     }
                  if( nx >= 2 && ny >= 2)
                     {
                     flag = 1;
                     knum1 = nx;
                     knum2 = ny;
                     break;
                     }
                   }
                   ++khz;
                }
                 break;
        case 't':
                 tic = atof(argv[1]);
                 ctick = CLK_TCK*tic;
                 break;
        case 'f':
                 /*if(argv[0][1] == '2')
                   {
                   opt = 2;
                   channel = twochan;
                   tmsfmt = totms2;
                   fftout = ftsize * 2;
                   }*/

                 if (argv[0][1] == '1')
                   {
                     if(argv[0][2] == 'e')
                       {
                       opt = 4;
                       channel = chanenv;
                       }
                     else if (argv[0][2] == 'b')
                       {
                       opt = 5;
                       channel = chanenv;
                       }
                     else
                       {
                       opt = 3;
                       channel = onechan;
                       }
                     fftout = ftsize;
                     tmsfmt = totms1;
                     }
                 break;
        case 'c':
                 if(argv[0][1] == 'c')
                 {
                     fp = fopen("colors.dat","r");
                     if(fp == NULL)
                        {
                         printf("Unable to open file Colors.dat.\n");
                         exit(-1);
                        }
                     else    /*colors will accept up to 16 values*/
                        {
                         n = 0;
```

```
                        while(fscanf(fp,"%d",&dat1) != EOF)
                            {
                            colors[n] = dat1;
                            n++;
                            }
                        fclose(fp);
                        }
                }
                else if (argv[0][1] == '1')
                    fp = fopen("levels.dat","r");
                    if(fp == NULL)
                      {
                        printf("Unable to open file Levels.dat.\n");
                        exit(-1);
                      }
                    else    /*edge will accept up to 16 values*/
                      {
                        n = 0;
                        while(fscanf(fp,"%x",&dat1) != EOF)
                            {
                            edge[n] = dat1;
                            n++;
                            }
                        fclose(fp);
                        }
                break;
            }
        }
        return(opt);
}

/*********************************************************************/
/*********************************************************************/
/*DASHIN.C
 dashin.c - routine to read up to two channels of DMA data. User may
            input parameters add1 and add2 to determine sample frequency
            through the runtime command line. On each restart after a
            memory display, the starting address is page 7, offset 0.
*/
/**********************************************************************/
#include <graph.h>
#include <stdio.h>
#include <conio.h>

#define BASE    0x310
#define MUX     BASE+2          /*used to establish number of channels*/
#define STATUS  BASE+8
#define CONTROL BASE+9          /*allows IRQ5 to be set*/
#define CTREN  BASE+10          /*counter enable*/
#define CTR0   BASE+12
#define CTR1   BASE+13
#define CTR2   BASE+14
#define CTRCONT BASE+15         /*counter control*/
#define DMACHANNEL    1
#define DMAMODE      0x45       /*01000101 for single mode select, address
                                   increment,NO auto reload of registers,write
                                   transfers,channel 1 select*/
#define BASEREG      2
#define COUNTREG    3
#define PAGEREG      0x83       /*DMA page register 1*/

/**********************************************************************/
```

```
dashin(buffer,chan)
 int chan;                    /*choice of number of channels (6/89 - max is 2)*/
 int *buffer;
 {
   int i,j, dmabasel, dmabaseh, dmacountl, dmacounth;
   int *statptr;
   extern int knum1,knum2;      /*for sampling frequency option*/
   extern int dmapage;
   extern int dmaknt;
   statptr = (int*)STATUS;
   dmaknt = 1;

/* set up the DMA parameters */
   dmabasel = 0;                /*start at the beginning of a page */
   dmabaseh = 0;
   dmapage = 7;                 /*this MUST be 7; pages = 7,8,9,4,5,6*/
   dmacountl =  0xff;
   dmacounth =  0xff;           /*transfer 64k pts  */
   outp(CTREN, 0);              /*disable data acquisition  */

/*program the DMA chip before starting DASH board*/
   outp(11, DMAMODE);
   outp(12, 0);
   outp(BASEREG, dmabasel);     /*start of memory address*/
   outp(BASEREG, dmabaseh);
   outp(COUNTREG, dmacountl);   /*number of bytes to transfer*/
   outp(COUNTREG, dmacounth);
   outp(PAGEREG, dmapage);      /*hereafter dmapage is rewritten in handle.asm*/
   outp(10, DMACHANNEL);

/*DASH board parameters*/
   outp(CONTROL, 0xD7);         /* enable interrupt 5 */
   outp(MUX, chan);             /* get 1 or 2 channels*/
   outp(CTRCONT, 0x74);         /* ctr 1, mode 0, write numcnt  */
   outp(CTR1, knum1);           /* for sampling frequency */
   outp(CTR1,  0);
   outp(CTRCONT, 0xb4);         /* ctr 1 divides by 4 */
   outp(CTR2, knum2);           /* for sampling frequency */
   outp(CTR2, 0);
   outp(CTREN, 01);             /* start data acquisition */
   handle();                    /* enable interrupt handler*/

   return(0);
}
/****************************************************************/

dashoff() {
   outp(CTREN,00);              /*turn off the counter enable */
   outp(0x0e, 0000);            /* reset the DMA chip */
   outp(0x0d, 0000);
   outp(0x0f, 0xff);
}
/*******************************************************************/
/*get the most recent address at which data has been received, and
  return this offset to use in address calculations. */

unsigned int dashget()
{
  unsigned i,j;

  j = ((inp(BASEREG)) | (inp(BASEREG) << 8));

  return(j);
}
```

```
/*********************************************************************/
/*********************************************************************/
/*DELMAG.C
delmag.c -- deletes clipping light; activated by key K_DEL
*/

#include <stdio.h>
#include <graph.h>

delmag()
{
 int i, txtcolor;

 txtprep();
 txtcolor = _gettextcolor();
 _settextcolor(_getbkcolor() );
 _settextwindow(18,61,28,62);

  for( i=1 ; i <= 10 ; i++ )
    {
     _settextposition(i,1);
     _outtext(" ");
    }

  _settextcolor( txtcolor );
}


/*********************************************************************/
/*********************************************************************/
;ENDINT5.ASM
;endint5.asm -- ends the interrupt condition invoked by handle.asm

EXTRN keep_cs:near
EXTRN keep_ip:near

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT

PUBLIC _endint5
_endint5  PROC FAR
        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di

        cli                             ;disables interrupts(int flag set to 1)
```

```
        push    ds
        mov     dx, WORD PTR keep_ip    ;prepare to restore offset
        mov     ax, WORD PTR keep_cs
        mov     ds, ax                  ;prepare to restore segment
        mov     ah, 25h                 ;function to set an interrupt vector
        mov     al, 0dh                 ;number of the vector
        int     21h                     ;now the vector is reset
        pop     ds                      ;restore ds
        sti                             ;clear flag to enable interrupts

        pop     di
        pop     si
        pop     ss
        mov     sp, bp
        pop     bp
        ret
_endint5        endp
_TEXT ENDS
END


/*******************************************************************/
/*******************************************************************/
;ERASE.ASM
;erase.asm
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)

;erases contents of bottom window (screen-width waveform)
;   erase(text column number)

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT
PUBLIC _erase

_erase PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
```

```
        out     dx, al
        mov     ax, 0a019h                  ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                    ;point to address register
        mov     al, 8                       ;bit mask register
        out     dx, al                      ;address the register
        inc     dx                          ;point to data register
        mov     ax, 80h                     ;mask out all bits except bit 7
        out     dx, al                      ;send data to mask register

;get the bar height to be plotted
        mov     ax, 128
        mov     dx, 80
        mul     dx
        add     ax, 10240
        add     ax, [bp+6]
        mov     dx, ax

;  put the color into the mask register
        mov     dx, 0                       ;color is black

;get the column number where cursor is to be written
        mov cx, [bp+6]
        mov bx, cx                      ;load with screen column number to write to
        add bx, 10240                   ;top of bottom window

;draw a pixel

col1:   mov al, es:[bx]                 ;fill the latch registers
        mov es:[bx], al                 ;draw the pixel
        mov byte ptr es:[bx], 00
        add bx, 80                      ;point to pixel below
        cmp bx, dx
        jl col1

colb:   mov  al, dl                      ;pixel color
        mov byte ptr es:[bx], 00
        add bx, 80                      ;point to pixel below
        cmp bx, 20560                   ;-- column bottom
        jl colb

        pop  ds
        pop di
        pop si
        pop ss
        pop bp
        ret

_erase  ENDP

_TEXT   ENDS

        END


/*********************************************************************/
/*********************************************************************/
/*GETMEM.C
  getmem.c --
  Sets up for the display of data stored in memory by the
  DMA in a buffer up to 384k large. Start-and-end addresses (page-offset)
  and count of pages of input data are calculated; then a display loop
```

```
  increments the input addresses and calls showdat to display the processed
  data at a resolution calculated to exactly fill the screen window.
  NB: The DMA controller has been stopped in the calling program. After
  each examination of memory, it is restarted by dashin() with page set
  to 7, offset set to 0.

  Time ticks are displayed at the top of the window for every second,
  with every tenth second in double length; if window width contains
  less than 4 seconds of data, every tenth of a second is marked with
  a half-length tick. NB: The time ticks in the zoom displays reflect
  the time span required for display of the selected data in real time.

  Operations available:

    zooms - sequential
    redisplay of prior zoom screens (up to 10 levels)
    save delimited data to file
    zoom; hit 's' key to save full screen of data
    move cursors; save data; hit enter to view full screen of saved data
    move cursors; save data; move cursors; hit enter to see subset of saved data
    move cursors; save data; move cursors; save data
      note: after a 'save', the cursors may be moved either toward the
            screen center or toward the side margins
*/
/**********************************************************************/

#include <stdio.h>
#include <graph.h>
#include <stdlib.h>

#include "keydefs.h"
#include "fcntl.h"
#include "hsmos.def"

#define O_RAW O_BINARY
#define enable_p1   wstcr( (rstcr()&STC_RUN) | STC_DPAGE1 )
#define disable_edm wstcr( rstcr()&STC_MEM_MASK )
#define enable_p0   wstcr( (rstcr()&STC_RUN) | STC_DPAGE0 )

#define WHITE 319
#define BLACK  256
#define WIDE 828

unsigned int dashget();
unsigned int showdat();
unsigned int (*move)();
void putcurs();
int getkey();

/**********************************************************************/

getmem(choice,jump)
   int choice;  /*display option set in cmdopt (# of channels) - needed
                  to funnel into showdat.c*/
   int jump;    /*initial offset increment factor */
{
  extern char curcolor;
  extern unsigned int doffset, doffinc, endoff;
  extern long unsigned int showaddr;
  extern int dmapage, knum1, knum2, lcol, rcol, cycle, *absptr;
  extern int inkey; /*value is established by calling function*/
  extern int dmaknt; /*initialized to 1 in dashin*/
  extern int gain;
  extern float tic;
```

```
/*local variables*/
static unsigned int strtpage, lstpage, firstoff, lastoff, oldstep;
static int kounter, pushknt;
static struct pushpop
 {
  unsigned int pge;
  unsigned int incr;
  unsigned int ofset;
  unsigned int offlast;
  int pgknt;
  }levels[10],*which;

double sfreq;
unsigned int step, pagenow, pagelast, strtbyte, endbyte;
unsigned int oldoff, tmp1, tmp2, tmp3, offseta;
int txtcolor, location, movknt, firstcol, lastcol;
int i, j, mark, knt, loopno, kt, sek, again, reply;
int tenloop, tenth, deci, kten, place, markten;

/*initializing*/
 deci = tenth = 0;
 place = tenloop = 1;
 reply = 0;
 firstcol = 0;
 lastcol = 60;
 loopno = 0;
 kten = kt = 1;
 again = 6;                    /*default for 6 pages of raw data*/
 oldoff = 0;
 sek = 1;                      /*time ticks -- either 1 or 0 */
 step = (unsigned)jump;
 sfreq = (1000000./((double)(knum1 * knum2)));   /*herz*/
 mark = (int)(((sfreq*2.0)/(float)step) * tic);  /*timeticks calculation*/

/****************************************************************/

/*****define the number of pages to be read and displayed, and
       the page and offset where data retrieval is to begin******/
/*
Variables to be defined:
     again  -  count of pages to be read (NB: pages are 4,5,6,7,8,9
               with beginning page containing data for the last page,
               so that 'again' for 6 pages = 6 for 7 iterations, 0-6)
     pagenow - starting page
     pagelast- ending page
     doffset - starting offset
     endoff  - last offset
*/

  switch (inkey) {
   case K_N:
    /*This case is ALWAYS called first, and sets up the basis for
      variable values in the rest of the function*/

        /*get current page & offset values*/
        pagelast = (unsigned)dmapage;
        offseta = dashget();
        endoff = offseta;

        if(dmaknt >= 7)
           /*normal situation -- a full 6 pages of brand new data*/
          {
           again = 6; /*loop control, 0 - 6 */
```

```
          pagenow = pagelast;
          doffset = offseta;
          step = step & 0xfffc; /*evenly divisible by 4*/
         }
       else
        /*less than 6 pages of new data. Either
          A) the user has hit the 'm' key so soon after the previous
             retrieval that the DMA has not been able to refill all
             six pages, or
          B) this is the start of the session, and 6 pp have not yet
             been filled.   */
         {
           again = dmaknt-1;
           pagenow = 7;
           doffset = 0;

         /*must redefine 'step' so that display fills the screen; step must be
           less than (138*6)*/
           step = (138 * again) + ((offseta)/480);
           step = step & 0xfffc; /*evenly divisible by 4*/
           doffinc = step;
         }

        /*save these parameters for subsequent displays of this memory*/
        pushknt = 0; /*initializes at each exit from running display*/
        which = &levels[pushknt];
        which->pge     = pagenow;
        which->incr    = step;
        which->ofset   = doffset;
        which->offlast = endoff;
        which->pgknt   = again;

    break;

  case K_ESC:
      /* display data from the previous screen */
      if( pushknt > 0 )
        {
         which = &levels[pushknt-1];
         --pushknt;
         if( pushknt == 0 ) messages(4);
         }
       else
        {
         which = &levels[pushknt];
         messages(4);
         }
         pagenow = which->pge;
         step    = which->incr;
         doffset = which->ofset;
         endoff  = which->offlast;
         again   = which->pgknt;
    break;
  default:
    /* this takes care of inkeys (set inside lcursor), which may be
       <enter> "zoom" or <s> "save to file" */

   /*Here we adjust address and page kount to match cursor moves.
     For iterative zooms, page and offset are calculated from the
     start of the previous zoom, NOT from start of buffer*/

       /*defaults*/
       strtbyte = firstoff;
       endbyte = lastoff;
```

```
        pagenow = strtpage;
        pagelast = lstpage;
        again = kounter;

        movknt = lcol;/* NB: 'movknt' is the number of cursor moves */
        again = movtorit(&strtbyte,again,oldstep,movknt,&pagenow);

        movknt = 60 - rcol;
        again = movtolft(&endbyte,again,oldstep,movknt,&pagelast);

        doffset = strtbyte;
        endoff = endbyte;

       /*find the incremental step to be used, and (maybe) store
         all the screen display parameters which have been newly
         calculated in "levels" */
           step = calcstep(again);
           step = step & 0xfffc;  /*evenly divisible by 4*/

           /*if step increment is < 20, there are rounding up or down
             problems so that the resulting screen display is not valid.
             However, a "save" of data is still possible because the
             doffset and endoff will be correct. */

           if( inkey != K_S )
             {
              if( step < 20)
                {
                 reply = messages(2);
                 if( reply == 1 )
                   inkey = K_S;
                 else
                  {
                   messages(9);
                   /*pick up values from previous display*/
                   doffset = firstoff;
                   endoff = lastoff;
                   pagenow = strtpage;
                   again = kounter;
                   step = oldstep;
                   }
                 }
              else
                {
                 /*stuff zoom parameters into structure array so
                   they will be available for the redisplay option*/
                 if( ++pushknt > 9 ) pushknt = 1;
                 which = &levels[pushknt];
                 which->pge     = pagenow;
                 which->incr    = step;
                 which->ofset   = doffset;
                 which->offlast = endoff;
                 which->pgknt   = again;
                 }
              }

 } /*end of switch*/

   doffset = doffset & 0xfffc; /*evenly divisible by 4*/

/***************End of defining addresses and page counts**************/

/**************************************************************************/
  /*Saving to file*/
```

```
   if( inkey == K_S )
     {
      delmag();  /*delete clipping light*/
      savscr(pagenow,doffset,endoff,again,step);
      dmaknt = 1;
      return(0);
      }
  /*do NOT save any of the parameters below*/

/************************************************************************/
   /*Save some static variable values for the next call to this
    function. Done here because pagenow and doffset are
    different at bottom of Display Loop from at top /

  firstoff = doffset;
  lastoff  = endoff;
  strtpage = pagenow;
  lstpage  = pagelast;
  kounter  = again;
  oldstep  = step;

/***********************************************************************
*******************Start of Display Loop*****************************/

      /*Loop from starting place in first page until:
            the same place is found, for a full 6 pp
                          or
            last offset (endoff) on last page is hit ( < 6 pp ).*/

  /*****************erase the current cursors****************/
      if( inkey != K_M )
              {
              location = lcol;
              putcurs(location);
              location = rcol;
              putcurs(location);
              }
  /***********************************************************/

       /*some initializing. . . */
       showaddr = ((long)pagenow)<<28;
       if( again < 0 ) again = 0;
       sek = 0;
       oldoff = 0;
       cycle = 0;
       mark = (int)(((sfreq*2.0)/(float)step) * tic);
       if( mark >= 120 ) tenth = 1; /*show tenth-of-second ticks*/

       /*here we go! */
       for( i=0 ; i<=again ; i++)
        {
         /*do this once to be sure doffset set > 0 */

             /*Making a time tick*/
             if(++loopno == (mark*kt) )
              {
               place = mark * kt;
               sek = 1;  /*make a time tick on the next loop*/
               ++kt;
               kten = 1;
               tenloop = loopno;
              }
```

```
    showdat(choice,sek,deci);
    sek = 0;
    oldoff = doffset;
    doffset += step;

while (doffset > oldoff)
 {
   if( kbhit() )
    {
     inkey = getkey();
     switch (inkey)
      {
      case K_X:
         goto done;     /*exit from voice*/
      case K_H:
         helpvce(2);    /*help window*/
         break;
      case K_DEL:
         delmag();      /*delete clipping light*/
         break;
      case K_UP:
         ++gain;
         break;
      case K_DOWN:
         --gain;
         break;
      default:
         messages(1);  /*legal keys in message center*/
         break;
      }
    }

   /*Making a time tick*/
   if(++loopno == (mark*kt) )
    {
     place = mark * kt;
     sek = 1;  /*make a one-second time tick*/
     ++kt;
     kten = 1;
     tenloop = loopno;
    }
   else
    {
     if( tenth )
       {
       deci = 0;
       markten = mark/10;
       if(( mark%10 ) > 5 ) ++markten;
       if( (++tenloop) == ((markten*kten) + (place) ) )
        {
         if( kten <= 9 ) deci = 1;
         ++kten;
         }
       }
     }
   showdat(choice,sek,deci);

   /*Set up for the next loop on this page*/
   deci = 0;
   sek = 0;
   oldoff = doffset;
   doffset += step;
   if((i == again) && (doffset >= endoff))
     {
```

```
                goto done;
              }
            }
          /*Set up for processing the next page*/
          oldoff = 0;
          ++pagenow;
          if( pagenow > 9) pagenow = 4;
          showaddr = ((long)pagenow)<<28;
           }

done:
  if( cycle > 0) movefull();  /*show the last scrap of data*/
  clearmsg();  /*erase any messages*/
  dmaknt = 1;  /*get ready for the next call to 'handle' DMA acquisition*/

 /**************draw the first and last cursors************************/
     curcolor = WHITE;
     putcurs(firstcol);
     putcurs(lastcol);

}

/******************************************************************/
/******************************************************************/
;HANDLE.ASM
;
;handles the DMA end-of-page interrupt

extrn _dmapage:WORD
extrn _dmaknt:WORD

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
keep_cs dw      0               ;holds segment for replaced interrupt
keep_ip dw      0               ;holds offset for replaced interrupt

_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT

PUBLIC  keep_cs
PUBLIC  keep_ip
PUBLIC _handle

_handle  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di

;Set up to receive interrupt
```

```
        mov     ah, 35h                 ;function to get int address
        mov     al, 0dh                 ;number of the vector
        int     21h                     ;now segment is in ES, offset in BX
        mov     keep_ip, bx             ;store offset
        mov     keep_cs, es             ;store segment

        push    ds                      ;save ds
        cli                             ;disable interrupts
        in      al,21h                  ;enable interrupt
        and     al,0dfh
        out     21h,al

        mov     dx, offset master       ;offset of interrupt routine in dx
        mov     ax, seg master          ;segment of the interrupt routine
        mov     ds, ax                  ;place in ds
        mov     ah, 25h                 ;function to set up a vector
        mov     al, 0dh                 ;the vector number (IRQ5)
        int     21h                     ;change the interrupt
        sti                             ;reenable interrupts
        pop     ds                      ;restore ds

        mov     dx,318h                 ;write to DASH status register
        xor     al,al
        out     dx,al

        pop     di
        pop     si
        pop     ss
        mov     sp,bp
        pop     bp

        ret                             ;go back to C calling routine
                                        ;with interrupt still enabled

_handle ENDP

;*********************************************************

;The interrupt routine...i.e., what to do when an interrupt is found
;dmaknt added to keep track of how many memory pp have been written to
;since the last access by getmem.c

master proc far

        push    bp
        push    ax
        push    cx
        push    dx
        push    bx
        push    es
        push    ss
        push    ds
        push    si
        push    di
        mov     bp,sp

        mov     dx, 0ah                 ;clears chanl 1 mask register
        mov     al,1                    ;bit in 8237A chip
        out     dx, al

;we must go through all these tricks to capture the value of dmapage
;because it is a global variable in memory, and because here we are
;a large (far) model
        mov     ax, seg _dmapage        ;get global dmapage from memory
```

```
        mov     es,ax                   ;put segment into es
        mov     bx, offset _dmapage     ;get address
        mov     ax,es:[bx]              ;get the value of dmapage

;redefine _dmapage to be ready for write
        xor     ah,ah
        inc     al
        cmp     al,9
        jle     next
        mov     al,4
next:
        mov     es:[bx],ax

;write dmapage to DASH pageregister port
        xor     ah,ah
        mov     dx, 083h                ;this gets PAGEREG into dx
        out     dx,ax                   ;write dmapage to pagereg

        mov     dx,0318h                ;write to DASH status register
        xor     al,al                   ;so that data can be acquired
        out     dx,al

;increment value of global dmaknt
        mov     ax, seg _dmaknt         ;get global dmaknt from memory
        mov     es,ax                   ;put segment into es
        mov     bx, offset _dmaknt      ;get address
        mov     ax,es:[bx]              ;get the value of dmaknt

        inc     al                      ;increment
        mov     es:[bx],ax              ;  and update the variable

;end of hardware interrupt
        mov     al, 020h                ;required for completion
        out     20h, al                 ;of hardware interrupts

        mov     sp,bp
        pop     di
        pop     si
        pop     ds
        pop     ss
        pop     es
        pop     bx
        pop     dx
        pop     cx
        pop     ax
        pop     bp

        iret
master endp

_TEXT   ENDS

        END


/**********************************************************************/
/**********************************************************************/
/*HELPVCE.C
helpvce.c -- "help" menus to be displayed when the 'h' key is hit.
*/

#include <stdio.h>
#include <graph.h>
```

```
helpvce(pick)
 int pick;
{
  int txtcolor, abbrvcol;

  _setcolor(63);
  _setcliprgn(500,138,639,260);
  _rectangle(_GBORDER,500,138,639,260);

  txtprep();
  txtcolor = _gettextcolor();
  abbrvcol = 5;
  _settextcolor(txtcolor);
  _settextwindow(11,64,20,80);

  if( pick==1 )
    {
     _settextposition(1,1);
     _outtext("   HELP MENU");

     _settextposition(3,1);
     _settextcolor(abbrvcol);
     _outtext("f");
     _settextcolor(txtcolor);
     _outtext("        freeze");

     _settextposition(4,1);
     _settextcolor(abbrvcol);
     _outtext("m");
     _settextcolor(txtcolor);
     _outtext("    see memory");

     _settextposition(5,1);
     _settextcolor(abbrvcol);
     _outtext("x");
     _settextcolor(txtcolor);
     _outtext("          exit");

     _settextposition(6,1);
     _settextcolor(abbrvcol);
     _outtext("del");
     _settextcolor(txtcolor);
     _outtext("   erase bar");
    }
  else if( pick == 2 )
    {
     _settextposition(1,1);
     _settextcolor(abbrvcol);
     _outtext("s");
     _settextcolor(txtcolor);
     _outtext("   save to file");

     _settextposition(2,1);
     _settextcolor(abbrvcol);
     _outtext("x");
     _settextcolor(txtcolor);
     _outtext("           exit");

     _settextposition(3,1);
     _settextcolor(abbrvcol);
     _outtext("F1,F2");
     _settextcolor(txtcolor);
     _outtext("  left curs");
```

```
    _settextposition(4,1);
    _settextcolor(abbrvcol);
    _outtext("F3,F4");
    _settextcolor(txtcolor);
    _outtext(" right curs");

    _settextposition(5,1);
    _settextcolor(abbrvcol);
    _outtext("esc");
    _settextcolor(txtcolor);
    _outtext(" prior screen");

    _settextposition(6,1);
    _settextcolor(abbrvcol);
    _outtext("del");
    _settextcolor(txtcolor);
    _outtext("    erase bar");

    _settextposition(7,1);
    _settextcolor(abbrvcol);
    _outtext("enter");
    _settextcolor(txtcolor);
    _outtext("  next zoom");

    _settextposition(8,1);
    _settextcolor(abbrvcol);
    _outtext("space");
    _settextcolor(txtcolor);
    _outtext("   realtime");
   }

}

/***********************************************************************/
/*clearhlp.c
  Totally erase top level help menu area, border & lines 11-20,
  columns 64-80.
*/

#include <graph.h>

clearhlp()
 {
  int txtcolor, i;
  _setcolor(0);
  _rectangle(_GBORDER,500,138,639,260);

  txtprep();
  txtcolor = _gettextcolor();
  _settextcolor(_getbkcolor() );
  _settextwindow(11,64,20,80);

   for( i = 1; i < 9; i++ )
     {
         _settextposition(i,1);
         _outtext("xxxxxxxxxxxxxxxx");
      }

   _settextcolor(txtcolor);
}

/*********************************************************************/
/*********************************************************************/
/*KAYHDR.C
```

```
 kayhdr.c -- Writes headers to output files of raw spectrogram data so
            that said files can be read by the Kay Sona-Graph
            Workstation, format 5500.
*/
/*************************************************************************/

#include <stdio.h>
#include <stdlib.h>

kayhdr()
 {
 extern FILE *stream;
 extern int knum1, knum2;

 int tmp0, tmp1, temp, i, j;
 unsigned int hertz;
 long int place;                      /*fseek requires this to be long*/
 float khertz, tmphertz;
 tmp0 = 0; tmp1 = 1;
 khertz = (1000./((double)(knum1*knum2)));
 tmphertz = (unsigned int)khertz * 1000.;
 hertz = tmphertz / 10.;

/*write 512 zeros to push EOF forward before starting a 'seek' */
    for( i = 0 ; i < 512 ; i++)
      {
      place = (long)i;
      fseek( stream,place,SEEK_SET);
      putw(tmp0,stream);
      }
    rewind(stream);

    place = 24;
    fseek( stream,place,SEEK_SET);
    fputs("12",stream);        /*bytes 25-26*/

    place = 38;
    fseek( stream,place,SEEK_SET);
    putw(tmp0,stream);         /*bytes 39-40*/

    place = 64;
    fseek( stream,place,SEEK_SET);
    fputs("5500SD",stream);    /*bytes 65-70*/

    place = 70;
    fseek( stream,place,SEEK_SET);
    putw( (long)tmp1,stream); /*bytes 71-74*/

    place = 120;
    fseek( stream,place,SEEK_SET);
    putw(tmp1,stream);          /*bytes 121-122*/

    place = 122;
    fseek( stream,place,SEEK_SET);
    putw(hertz,stream);         /*bytes 123-124*/

    place = 124;
    fseek( stream,place,SEEK_SET);
    temp = -32000;
    putw(temp,stream);          /*bytes 125-126*/

    place = 126;
    fseek( stream,place,SEEK_SET);
    temp = 32149;
```

```
    putw(temp,stream);          /*bytes 127-128*/

  /*Bytes 129 - 150 have already been filled with zeroes; these are the
    fields for "spectral" data; we are saving "sample" data instead*/

  /*Space forward to byte 512, where data will begin*/

    place = 512;
    fseek( stream,place,SEEK_SET);

    return(0);
 }


/*******************************************************************/
/*******************************************************************/
/*KEYS.C
 keys.c -- Functions to read keystrokes, and to determine actions to be
           taken, depending on which keys are hit.
*/
/**********************************************************************/

#include "keydefs.h"

/**********************************************************************/

/*Getkey -- returns code for single combo keystrokes
    unique code for each keystroke or combination
*/

#include <dos.h>

#define KEYIN  0x7
#define LOBYTE 0x00FF

/**********************************************************************/

int getkey()
{
  int ch;

/*normal key codes*/
  if((ch = bdos(KEYIN, 0, 0) & LOBYTE) != '\0')
     return(ch);

/*convert scan codes to unique internal codes*/
     return((bdos(KEYIN, 0, 0) & LOBYTE) | XF);
}
/***********************************************************************/

/* keyopts.c -- key options which control progress of Voice during
                execution.
*/
/***********************************************************************/

#include <stdio.h>
#include <graph.h>
#include <conio.h>

#define EXWIDE  (138*6)    /*divisible by both 2 and 4*/
#define WHITE 319
#define BLACK  256

void putcurs();
```

```
void delmag();
void lcursor();
unsigned int endint5();
int getkey();
int dashoff();

/***************************************************************************/

keyopts(chanknt,choice,form)
  int chanknt;          /*same as numchan in main */
  int choice;           /*display option set in cmdopt (# of channels)*/
  int form;             /*saved output data is raw data or colorcodes*/
{

extern unsigned int *absptr, doffinc;
extern long unsigned int absaddr;
extern int speed, gain, inkey, tknt, lcol, rcol;
extern char curcolor;

int txtcolor, location, i;
unsigned int incr;
unsigned int dashtmp = 0;

lcol = 0;
rcol = 60;
incr = EXWIDE;/*default*/
curcolor = WHITE;

        inkey = getkey();
        switch (inkey){
           case K_X:
quit:
              endint5();
              dashoff(); /* turn off the DMA loop */
              _clearscreen(_GCLEARSCREEN);
              _displaycursor( _GCURSORON );
              setscr();
              _setvideomode(_DEFAULTMODE);
              exit(0);
           case K_UP:
             ++gain;
             break;
           case K_DOWN:
             --gain;   /*allows negative gain*/
             break;
           case K_LEFT:
             if (speed > 256) speed -= 256;
             else speed = 0;
             break;
           case K_RIGHT:
             speed += 256;
             break;
           case K_DEL:
             delmag();   /*delete clipping light*/
             break;
           case K_F:
             endint5();  /*stop DMA*/
             dashoff();
             while( getkey() != K_SPACE );
             dashin(absptr,chanknt);  /*restart DMA*/
             tknt = 0;
             while( dashtmp < 512 )
               {
                dashtmp = dashget();
```

```
        }
      break;
    case K_H:
      helpvce(1);
      break;
    case K_M:
      endint5();
      dashoff();

      clearmsg();  /*erase existing message window*/
      clearhlp();  /*erase main help menu*/

      doffinc = EXWIDE;
      incr = doffinc;

      lcol = 0; rcol = 60;

      getmem(choice,incr);
      lcursor();

      /*NB: the following inkey values are entered in lcursor*/

      while( inkey != K_SPACE )
        {
         /* allow repeated zooms */
         while( inkey == K_RETURN )
          {
              getmem(choice,incr);/*calculate the start and
                                    end offsets and pages, and
                                    display the delimited signal*/
              lcol = 0; rcol = 60;
              lcursor();
            }

         /* allow repeated saves to file */
         while( inkey == K_S )
          {
              getmem(choice,incr); /*no new display*/
              lcursor();
           }

         /* send the previous display to the screen */
         while( inkey == K_ESC )
          {
              getmem(choice,incr);
              lcursor();
           }

        clearmsg()   /* erase contents of message center */

        if( inkey == K_X )
              goto quit;

       } /*end of !=K_SPACE loop*/

     /********reset at end of memory display(K_M) ************/

    /*restart the A/D board*/
       doffinc = EXWIDE;
       dashin(absptr,chanknt);
       tknt = 0;
          /*Do some introductory loops to prevent garbage data
            (retrieved from BEFORE the first save-buffer) from
            being displayed.*/
```

```
            while( dashtmp < 512 )
              {
               dashtmp = dashget();
              }

         /*erase the old cursors*/
         location = rcol;
         putcurs(location);

         location = lcol;
         putcurs(location);

       /*erase wide waveform*/
         if( choice == 5 )
            {
             for( i = 0 ; i < 60 ; i++ )
                 erase(i);
             delmag();
            }

       /*erase help display*/
         clearhlp();

       /*put a cursor at start of running display*/
         linecurs();
         break;

     default:
         messages(3);  /*display of legal keys*/
         break;

     } /*end of outside switch */
}


/*******************************************************************/
/*******************************************************************/
/*LABELV.C
     Graphics routines to establish and display annotations before
     start of spectrogram display:
       labelv
       boxes
       blkbox
       putft
*/
/********************************************************************/

#include <stdio.h>
#include <graph.h>

 int row,trow1,tcol1,x1,iy1;

/*******************************************************************/

/* labelv.c - writes labels to screen for main VOICE using Microsoft
   C version 5.0 graphics. There are 8 pixels/column, 14 pixels/row.

   note: the 4-channel display (disp = 1) is not yet implemented*/
/*******************************************************************/

labelv(disp)
int disp;  /*this is "option" in main, and is set in cmdopt */
```

```
{
 extern int edge[];
 extern int colors[];
 int i,j;
 int y2;
 int oldcolor, colortxt;
 int colk,boxk;              /*number of columns, number of boxes needed*/
 char tbuf[5];
 char bound[6];

 /*initialize for Microsoft graphics*/
  _setvideomode(_ERESCOLOR);  /* must do this before setting the palette*/

/***set up and display voltage color code boxes with annotations***/

  oldcolor = ( _getcolor() );  /*save the current default color*/
  chcolor(colors);             /*set up the color palette*/

  /*set up for alternate display screens*/
  if(disp != 1)                /*one and two channel displays*/
   {
    _setcliprgn(0,256,640,350);
    trow1 = 22;
    x1 = 2;
    iy1 = 294;
    tcol1 = 4;
    colk = 2;
    boxk = 4;
     }
  else
   {                           /*four-channel display*/
    _setcliprgn(504,0,640,350);
    trow1 = 15;
    x1 = 512;                  /* @ 8 pixels/column*/
    iy1 = 196;                 /* @ 14 pixels/row, this is row 14*/
    tcol1 = 68;
    colk = 1;
    boxk = 10;
     }

  /*establish the area where all these boxes will be written*/
  _settextwindow(trow1,tcol1,(trow1+boxk),(tcol1+8));

  /*Now make the boxes and annotate them . . . */

  /*first output a black box enclosed by a white border*/
    row = 1;
    blkbox(0);

  /*Next complete 3 more boxes in the first column*/
    for(i = 1 ; i < 4 ; i++)
      {
       _setcolor(i);
       y2 = iy1 + 11;
       _rectangle(_GFILLINTERIOR,x1,iy1,(x1+18),y2);
       _settextposition(i+1,2);
       itoa(edge[i],bound,10);
       _outtext( bound );
       iy1 = y2 + 3;
        }

  /*Do the remaining columns*/
    if(colk > 1)
      {
```

```
        x1 = 160;
        iy1= 294;
        tcol1 += 20;
        row = 1;
         }
      else
        row = 5;

      boxes(colk,boxk);

   /*Finish colorcode section by resetting the default background color*/
      _setcolor( oldcolor );

/*******************End of annotated boxes************************/

   /*set up text color*/
      colortxt = (10);
      _settextcolor( colortxt );

/*** Annotate the sample rate **/
   if(disp == 1)                 /* 4-sample display*/
      {
      _settextwindow(1,64,4,80);
      _settextposition(1,1);
      _outtext("Fmax =      kHz");
      }
   else                          /* 1 or 2-sample display*/
      {
       _settextwindow(1,64,10,80);/* Col 63 is first usable col for text*/
       _settextposition(1,1);
       _outtext("Fmax =      kHz");
       _settextposition(9,1);
       _outtext("Fmin = 0");
      }

/***set up a Message Center**/
       _setcolor(63);
       _rectangle(_GBORDER,500,265,639,349);

/* draw horizontal line between annotations and display area*/
       _moveto(1,265);
   _lineto(480,265);
 }

/*****************************************************************/

/*boxes.c - draw and annotate "colknt" columns of "bxknt" color
             codes in each column.
*/
/******************************************************************/

boxes(colknt,bxknt)

int colknt,bxknt;

{
 extern int row,x1,iy1,trow1,tcol1,edge[];
 int i,j,x2,y2,end;
 char bound[8];
 end = 0;

  for (j = 0 ; j < colknt ; j++)
     {
      end += 4;
```

```
    _settextwindow(trow1,tcol1,(trow1+bxknt),(tcol1+8));
    x2 = x1 + 18;
    for(i = end ; i < end+bxknt ; i++)
      {
       if(edge[i+1] <= 0) goto lastbox;
       y2 = iy1 + 11;
       _setcolor(i);
       _rectangle(_GFILLINTERIOR,x1,iy1,x2,y2);
       _settextposition(row,2);
       itoa(edge[i],bound,10);
       _outtext( bound );
       iy1 = y2 + 3;
       ++row;
       }
    x1 += 160;
    iy1 = 294;
    tcol1 += 20;
    row = 1;
    }
  lastbox:  blkbox(i);
}

/**********************************************************************/

/* blkbox.c -- outputs to screen a black cube outlined by white.
*/
/**********************************************************************/

 blkbox(edgeknt)
  int edgeknt;
{
  char bound[8];
  int blkbord;
  extern int row,trow1,tcol1,x1,iy1,edge[];

  _setcolor(63);          /*needs to be set to bright white for the border*/
  _rectangle(_GBORDER,x1,iy1,(x1+18),(iy1+11));
  _settextposition(row,2);
   itoa(edge[edgeknt],bound,10);
  _outtext( bound );
  iy1 += 14;
}
/************************************************************************/

/* putft.c -- writes value of frequency to specified location on screen.
*/
/************************************************************************/

putft(freq,secdif,disp)
 double secdif;
 float freq;
 int disp;
{
 char seconds[6],frqmax[6];
 int precision = 3;

/*fix up the screen for writing an MSC Graphics label*/
  txtprep();

  gcvt(freq,precision,frqmax);  /*convert arg1 from double to char*/

  /*put max frequency on the y axis*/
  _settextwindow(1,64,2,80);
  _settextposition(1,8);
```

```
  _outtext( frqmax );
}

/*******************************************************************/
/*******************************************************************/
/*LCURSOR.C
  lcursor.c -- writes 2 cursor lines bracketing halted running display.
              Allows user to shift cursors left and right.

  Function keys:
     F1  - left cursor moves left
     F2  - left cursor moves right
     F3  - right cursor moves left
     F4  - right cursor moves right

  Defaults:
     lcol = 0
     rcol = 60
*/
/*********************************************************************/

#include <stdio.h>
#include <graph.h>
#include "keydefs.h"

#define WHITE 319   /* this is 63 + 256 */
#define BLACK 256   /* this is  0 + 256 */

void putcurs();
int getkey();

/*********************************************************************/

lcursor()
{
 extern int lcol,rcol, inkey;
 extern char curcolor;
 int i, j, location, txtcolor, trapflg, pxcol;
 curcolor = WHITE;
 trapflg = 0;

  while(1)
  {
    if(kbhit() )
    {
      inkey = getkey();
      switch (inkey)
      {
        case K_F1:
          if(lcol > 0)
            {
             location = lcol;
             putlcurs(location);  /*erase current white cursor*/
             --lcol;
             location = lcol;
             putlcurs(location);  /*write white cursor in new position*/
            }
          break;
        case K_F2:
          location = lcol;
          if( lcol == 0 )
              putcurs(location);
          else
              putlcurs(location);
```

```
              if( ++lcol >= (rcol - 3) )
                  {
                   printf("\a");
                   --lcol;
                   }

              location = lcol;
              putlcurs(location);
              break;
            case K_F3:
              location = rcol;
              if( rcol == 60 )
                  putcurs(location);
              else
                  putrcurs(location);

              if( --rcol <= (lcol + 3) )
                  {
                   printf("\a");
                   ++rcol;
                    }

              location = rcol;
              putrcurs(location);
              break;
            case K_F4:
              if(rcol < 59)
                {
                 location = rcol;
                 putrcurs(location);
                 ++rcol;
                 location = rcol;
                 putrcurs(location);
                }
              break;
            case K_DEL:
              delmag();               /*delete clipping light below amplitude wave*/
              break;
            case K_H:
              helpvce(2);             /*write to the 'help' window*/
              break;
      case K_X:
              if( trapflg )
                  clearmsg();
              return;                 /*shut down the program*/
      case K_S:
              if( trapflg )
                  clearmsg();
              return;                 /*save delimited data to file*/
      case K_SPACE:
              if( trapflg )
                  clearmsg();
              return;                 /*return to running display*/
      case K_RETURN:
              if( trapflg )
                  clearmsg();
              return;                 /*zoom display the delimited data*/
      case K_ESC:
              if( trapflg )
                  clearmsg();
              return;                 /*show the previous screen display*/
            default:
              messages(1);            /*display legal keys in message center*/
```

```
            trapflg = 1;
            break;
          }
        }
      }
    }

/******************************************************************/

/*linecurs.c -- displays a vertical white line as a cursor.
  This version good for one-channel display only.
*/
/******************************************************************/

unsigned int scrntop();
unsigned int (*move)();

/******************************************************************/

linecurs()
{
  extern char color1[];
  extern int cycle;
  int i, j;

  for( j=0 ; j<128 ; j++)
     color1[j] = WHITE;

  scrntop(color1, cycle++);

  if(cycle >= 8)
   {
   cycle = 0;
   move();
   }

}


/*********************************************************************/
/*********************************************************************/
/*MESSAGES.C
 messages.c -- Functions to write to and clear messages from the Voice
               message center. Message box was created in function
               labelv.c, with pixel dimensions 500,265 to 639,349
               @ 8 pixels/col, 14 pixels/row.
*/
/*********************************************************************/

#include <graph.h>
#include <stdio.h>

/*********************************************************************/

messages(pick)
 int pick;
{
 extern char fidid[], newfid[];
 int reply, answer, txtcolor, nucolor, i;
 nucolor = 5;
 answer = 0;

  txtprep();
```

```
txtcolor = _gettextcolor();
_settextcolor(txtcolor);

_settextwindow(20,64,25,80);

clearmsg();                    /*get rid of any current messages*/
printf("\a");                  /*ring a bell*/

if( pick == 1 )
  {
  /*display names of keys legal during memory buffer display*/
    _settextposition(1,1);
    _outtext(" Legal keys:");
    _settextposition(2,2);
    _outtext("F1,F2,F3,F4");
    _settextposition(3,2);
    _outtext("x, s, h (help)");
    _settextposition(4,2);
    _outtext("<esc>,<del>");
    _settextposition(5,2);
    _outtext("<space>,<enter>");
   }

else if ( pick == 2 )
  {
  /*zoom limit warning*/
    _settextposition(1,1);
    _outtext("No zoom display.");
    _settextposition(3,1);
    _outtext("Filesave?(Y/N) ");
    _settextposition(4,1);
    _outtext(">>");
    reply = getch();
    if( reply == 'Y' || reply == 'y' )
        answer = 1;
   }

else if ( pick == 3 )
  {
  /*display names of keys legal during realtime display*/
    _settextposition(1,1);
    _outtext(" Legal keys:");
    _settextposition(3,2);
    _outtext("h (help)");
    _settextposition(4,2);
    _outtext("f, m, x,");
    _settextposition(5,2);
    _outtext("<del>");
  }

else if ( pick == 4 )
  {
  /*tell user he has popped back to original display of memory*/
    _settextposition(2,1);
    _outtext(" Primary memory ");
    _settextposition(3,1);
    _outtext(" display");
   }

else if ( pick == 5 )
  {
    /*for filesave I/O*/
    _settextposition(2,1);
    _outtext(" File ID? ");
```

```
      _settextposition(3,1);
      _outtext(" >>");
     }

else if ( pick == 6 )
   {
      /*for filesave I/O*/
      _settextposition(1,2);
      _outtext( newfid );
      _settextposition(2,1);
      _outtext( "This file");
      _settextposition(3,1);
      _outtext("contains data.");
      _settextposition(4,1);
      _outtext("Append? (Y/N)");
      _settextposition(5,1);
      _outtext(">>");
      reply = getch();
      if( reply == 'N' || reply == 'n' )
          answer = 1;
   }

else if ( pick == 7 )
   {
      /*for filesave I/O*/

   /*display the filename*/
      _settextposition(2,1);
      _outtext("The save file is");
      _settextposition(3,2);
      _outtext( newfid );

   /*give a progress report to the user*/
      _settextposition(5,1);
      _outtext("Save in progress");
     }

else if ( pick == 8 )
   {
      /*for filesave I/O*/

   /*display the filename*/
      _settextposition(1,2);
      _outtext( newfid );

    /*give a final progress report to the user*/
      _settextcolor(nucolor);
      _settextposition(5,1);
      _outtext("Save completed");
      _settextcolor(txtcolor);
   }

else if ( pick == 9 )
   {
      _settextposition(3,1);
      _outtext("Showing ");
      _settextposition(4,1);
      _outtext("previous display");
   }

else if ( pick == 10 )
   {
      _settextposition(2,1);
      _outtext( "This file");
```

```
      _settextposition(3,1);
      _outtext("contains data.");
      _settextposition(4,1);
      _outtext("Enter new file");
      _settextposition(5,1);
      _outtext(">>");
     }

  else if ( pick == 11 )
    {
      _settextposition(2,1);
      _outtext("No data saved");
     }

 return(answer);

}

/**************************************************************************/

/*clearmsg.c
  Clears out any messages written to screen in message area lines 20-25,
  columns 64-80.
*/
/**************************************************************************/

clearmsg()
 {
  int txtcolor, i;

  txtprep();
  txtcolor = _gettextcolor();
  _settextcolor(_getbkcolor() );
  _settextwindow(20,64,25,80);

   for( i=1 ; i<6 ; i++ )
     {
       _settextposition(i,1);
       _outtext("xxxxxxxxxxxxxxx");
       }

  _settextcolor(txtcolor);

}


/*********************************************************************/
/*********************************************************************/
;MOVE.ASM
; Moves an 8-pixel column of data (written at righthand side of display
; window) to the left, allowing more data to be written on the right
; without overwriting existing data.

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS
```

```
DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT

PUBLIC _movetop

_movetop  PROC FAR
        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     ax, 0a019h              ;video ram base
        mov     es, ax                  ;set up segment pointer
        mov     ds, ax

        mov     dx, 3ceh                ;set write mode 1
        mov     al, 5                   ;index register 5
        out     dx, al                  ;send the index
        inc     dx                      ;point to mode register
        mov     al, 1                   ;choose mode 1
        out     dx, al                  ;set the mode
        cld                             ;clear direction flag (autoinc. movs)

        mov     bx, 0                   ;bx points to row
        mov     dx, 61                  ;number of columns to move
        mov     bp, 80                  ;next row pointer
newrow:
        mov     di, bx                  ;di = destination
        mov     si, di
        inc     si                      ;si = source
        mov     cx, dx                  ;cx = column counter
        rep movsb                       ;move over 1 row
        add     bx, bp                  ;point to next row
        cmp     bx, 10240               ;done with all rows ?
        jle     newrow                  ;no, go do next row

        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret
_movetop   ENDP

PUBLIC _movefull

_movefull  PROC FAR
        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     ax, 0a019h              ;video ram base
        mov     es, ax                  ;set up segment pointer
        mov     ds, ax

        mov     dx, 3ceh                ;set write mode 1
```

```
        mov     al, 5                   ;index register 5
        out     dx, al                  ;send the index
        inc     dx                      ;point to mode register
        mov     al, 1                   ;choose mode 1
        out     dx, al                  ;set the mode
        cld                             ;clear direction flag (autoinc. movs)

        mov     bx, 0                   ;bx points to row
        mov     dx, 61                  ;number of columns to move
        mov     bp, 80                  ;next row pointer
again:
        mov     di, bx                  ;di = destination
        mov     si, di
        inc     si                      ;si = source
        mov     cx, dx                  ;cx = column counter
        rep movsb                       ;move over 1 row
        add     bx, bp                  ;point to next row
        cmp     bx, 20480               ;done with all rows ?
        jle     again                   ;no, go do next row

        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret
_movefull   ENDP
_TEXT   ENDS
        END


/*******************************************************************/
/*******************************************************************/
;PUTCURS.ASM
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)
;
;Write a vertical linecursor to the screen
;   putcurs(text column number)

extrn _curcolor:BYTE

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT
PUBLIC _putcurs

_putcurs PROC FAR

        push    bp
```

```
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     ax, seg _curcolor       ;pass segment to ds
        mov     ds, ax
        mov     bx, offset _curcolor    ;ds:[bx]


        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     ax, 80h                 ;mask out all bits except bit 7
        out     dx, al                  ;send data to mask register

;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al

;  put the color into the mask register
        mov     dx, ds:[bx]

;get the column number where cursor is to be written
        mov     cx, [bp+6]
        mov     bx, cx                  ;load with screen column number to write to

;*****************************************************
;draw a pixel
col1:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, dl                  ;pixel color
        mov     es:[bx], al             ;draw the pixel
        add     bx, 80                  ;point to pixel below
        cmp     bx, 20480               ;20480 for 256 pts -- column bottom
        jl      col1

;*****************************************************
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_putcurs   ENDP

_TEXT      ENDS
```

```
        END

/*******************************************************************/
/*******************************************************************/
;PUTLCURS.ASM
;
;Write a vertical linecursor to the screen at specified "text" column
;with the cursor appearing as a left-bracket

;argument --   putlcurs(text column number)

;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)

extrn _curcolor:BYTE

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT
PUBLIC _putlcurs

_putlcurs  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     ax, seg _curcolor       ;pass segment to ds
        mov     ds, ax
;get the column number where cursor is to be written
        mov     cx, [bp+6]
        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     ax, 255                 ;1111 1111 - mask sets cursor width
        out     dx, al                  ;send data to mask register
```

```
;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al

;  put the color into the mask register
        mov     bx, offset _curcolor
        mov     dx, ds:[bx]

;load screen column number
        mov     bx, cx

; draw the first pixel row in the column
        mov     al,es:[bx]              ;fill the latch registers
        mov     al, dl                  ;pixel color
        mov     es:[bx], al             ;draw the pixel

;*****************************************************************
; now redefine the mask, and draw the rest of the column
;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     ax, 128                 ;1000 0000 - mask sets cursor width
        out     dx, al                  ;send data to mask register

;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al

;  put the color into the mask register
        mov     bx, offset _curcolor
        mov     dx, ds:[bx]
;load screen column number
        mov     bx, cx

;****************************************************
;draw a pixel

col1:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, dl                  ;pixel color
        mov     es:[bx], al             ;draw the pixel
        add     bx, 80                  ;point to pixel below
        cmp     bx, 20400               ;column bottom
        jl      col1

;****************************************************
; finally, redefine mask and draw the last pixel row

        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     ax, 255                 ;1111 1111 - mask sets cursor width
        out     dx, al                  ;send data to mask register
```

```
;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al

;  put the color into the mask register
        mov     bx, offset _curcolor
        mov     dx, ds:[bx]
;load screen column number
        mov     bx, cx
        add     bx, 20400

; draw the last pixel row in the column
        mov     al,es:[bx]              ;fill the latch registers
        mov     al, dl                  ;pixel color
        mov     es:[bx], al             ;draw the pixel

;*****************************************************

        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_putlcurs   ENDP

_TEXT ENDS

        END


/*******************************************************************/
/*******************************************************************/
;PUTMAG.ASM
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)
;

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
value equ 80h
_DATA   ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT

PUBLIC _putmag
```

```
_putmag PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     di, [bp+6]              ;pass offset of data buffer
        mov     ax, [bp+8]              ;pass segment to ds
        mov     ds, ax
        mov     si, 80h                 ;mask register

        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
setup:  mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     cx, [bp+10]             ;load column number from call variable
        mov     ax, 80h                 ;load value to be shifted
        shr     ax, cl                  ;compute bit mask for proper column
        out     dx, al                  ;send bit mask to mask register

;get the bar height to be plotted

;plotting a symmetric envelope - symmetry about a centerline
;plot black bar, followed by a color bar with width symmetric about the
;middle, finished off with another black bar

        mov     ax, 64                  ;load max height
        sub     ax, ds:[di]             ;subtract bar height
        mov     dx, 80
        mul     dx                      ;get row number to start
        add     ax, 10240               ;locate below spectral window
        add     ax, 60                  ;go to column 60
        mov     cx, ax                  ;cx holds the address for top boundary

        mov     ax, 64                  ;do again for bottom boundary
        add     ax, ds:[di]             ;add bar height
        mov     dx, 80
        mul     dx                      ;get row number to start
        add     ax, 10240               ;locate below spectral window
        add     ax, 60                  ;go to column 60
        mov     dx, ax                  ;dx holds the address for bottom boundary

        mov     bx, 60+10240            ;address to start on(60 cols,2*128 rows)
col1:   mov     al, es:[bx]             ;fill the latch registers
        mov     byte ptr es:[bx], 00    ;draw the pixel
        add     bx, 80                  ;point to pixel below
        cmp     bx, cx                  ;are we at the threshold
        jl      col1

col2:   mov     al, es:[bx]             ;load color 4 for lower part
        mov     byte ptr es:[bx], 04
```

```
        add     bx, 80
        cmp     bx, dx                  ;done with column?
        jl      col2

col3:   mov     al, es:[bx]             ;load color 4 for lower part
        mov     byte ptr es:[bx], 00
        add     bx, 80
        cmp     bx, 20480               ;done with column?
        jl      col3

finish:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_putmag   ENDP

_TEXT     ENDS

          END


/*********************************************************************/
/*********************************************************************/
;PUTRCURS.ASM
;
;Write a vertical linecursor to the screen at specified "text" column
;with the cursor appearing as a right-bracket
;
;argument --   putrcurs(text column number)
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)

extrn _curcolor:BYTE

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT
PUBLIC _putrcurs

_putrcurs  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
```

```
        push    ds

        mov     ax, seg _curcolor           ;pass segment to ds
        mov     ds, ax
;get the column number where cursor is to be written
        mov     cx, [bp+6]
        mov     dx, 3ceh                    ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                       ;video mode 2
        out     dx, al
        mov     ax, 0a019h                  ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                    ;point to address register
        mov     al, 8                       ;bit mask register
        out     dx, al                      ;address the register
        inc     dx                          ;point to data register
        mov     ax, 255                     ;1111 1111 - mask sets cursor width
        out     dx, al                      ;send data to mask register

;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al

;  put the color into the mask register
        mov     bx, offset _curcolor
        mov     dx, ds:[bx]

;load screen column number
;     need one column to the left for bracket to point left
        dec     cx
        mov     bx, cx

; draw the first pixel row in the column
        mov     al,es:[bx]                  ;fill the latch registers
        mov     al, dl                      ;pixel color
        mov     es:[bx], al                 ;draw the pixel

;*******************************************************************
; now redefine the mask, and draw the rest of the column

;set up bit mask register
        mov     dx, 3ceh                    ;point to address register
        mov     al, 8                       ;bit mask register
        out     dx, al                      ;address the register
        inc     dx                          ;point to data register
        mov     ax, 128                     ;1000 0000 - mask sets cursor width
        out     dx, al                      ;send data to mask register

;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al
```

```
; put the color into the mask register
        mov     bx, offset _curcolor
        mov     dx, ds:[bx]
;load screen column number
        inc     cx                      ;to return to valid column number
        mov     bx, cx

;*******************************************************
;draw a pixel

col1:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, dl                  ;pixel color
        mov     es:[bx], al             ;draw the pixel
        add     bx, 80                  ;point to pixel below
        cmp     bx, 20400               ;20480 for 256 pts (column bottom)
        jl      col1

;*******************************************************
; finally, redefine mask and draw the last pixel row

        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     ax, 255                 ;1111 1111 - mask sets cursor width
        out     dx, al                  ;send data to mask register

;XOR the cursor color so that the screen data can be redisplayed later
        dec     dx
        mov     al, 03h
        out     dx, al
        inc     dx
        mov     al, 018h
        out     dx, al

; put the color into the mask register
        mov     bx, offset _curcolor
        mov     dx, ds:[bx]
;load screen column number
;       need one column to the left for bracket to point left
        dec     cx
        mov     bx, cx
        add     bx, 20400

; draw the last pixel row in the column
        mov     al,es:[bx]              ;fill the latch registers
        mov     al, dl                  ;pixel color
        mov     es:[bx], al             ;draw the pixel

;*******************************************************

        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_putrcurs   ENDP

_TEXT ENDS
END

/*********************************************************************/
```

```
/**********************************************************************/
/*REVIEW.C
  review.c -- displays a file of binary data which has been
  saved during previous runs of Voice. Input file may be any size;
  it is read in chunks of 512 bytes.

  August 1989 version allows the display of one channel of data, invoked
  by including -r and the filename on the command line.
*/
/**********************************************************************/

#include <stdio.h>
#include <graph.h>

#include "keydefs.h"

#include "fcntl.h"
#include "hsmos.def"
#define O_RAW O_BINARY
#define enable_p1   wstcr( (rstcr()&STC_RUN) | STC_DPAGE1 )
#define disable_edm wstcr( rstcr()&STC_MEM_MASK )
#define enable_p0   wstcr( (rstcr()&STC_RUN) | STC_DPAGE0 )

unsigned int scrntop();
unsigned int scrnbot();
unsigned int totms1();
unsigned int totms2();
unsigned int movetop();
unsigned int movefull();
unsigned int (*move)(),(*tmsfmt)();

/**********************************************************************/

review(disp)
  int disp; /*'option' in main*/
 {
   extern FILE *stream;
   extern char fidin[], color1[], color2[];
   extern unsigned int *dataptr, *tmsptr;
   extern unsigned int doffinc; /* default = 138*6 */
   extern long unsigned int tmsaddr;
   extern int inkey, gain, cycle, fftval[], *lp1, *lp2, *ltmp, lcol, rcol;
   extern int ptknt, fftout, kay;
   FILE *fp;
   int i,j,k,m,n,loopknt;
   unsigned int revdata[256],revknt;
   unsigned int rawval;
   long int strt, place;
   revknt = cycle = 0;
   gain = 0;

/*open the data input file*/
    if((fp = fopen(fidin,"rb")) == NULL)
      {
      _clearscreen(_GCLEARSCREEN);
      _displaycursor( _GCURSORON );
      _setvideomode(_DEFAULTMODE);
      setscr();
      printf("Could not open input file\n");
      exit(0);
      }

/*read data input file in 512-byte chunks; process and output each chunk*/
     if( kay )
```

```
        {
          place = 512;
          fseek(fp,place,SEEK_SET);
         }
      strt = ftell(fp);

/* Top of user-control and read-execute loop*/
   while(1){
      if(kbhit() ){
        inkey = getkey();
        switch (inkey){
           case K_F:
              while(!kbhit());
              break;
           case K_S:
               lcursor();
               savscr(1);
               /*move fp forward the required number of bytes*/
               if( lcol > 1)
                 fseek(fp,(long)(((lcol-1)*8) * 272),SEEK_CUR);
              fwrite(fp,sizeof(int),((rcol-lcol+1) * 4 * 136),stream);
              break;
           case K_X:
              fclose(fp);
              _clearscreen(_GCLEARSCREEN);
              _displaycursor( _GCURSORON );
              setscr();
              _setvideomode(_DEFAULTMODE);
              exit(0);
          case K_UP:
             ++gain;
             break;
          case K_DOWN:
             if (gain > 0) --gain;
             break;
          }
      }
      for( n=0 ; n < 256 ; n++)
         if( kay )
           {
            rawval = getw(fp);
            revdata[n] = rawval << 4;
            }
         else
           {
            revdata[n] = getw(fp);
            }
      revknt++;   /*tally of 512-byte chunks*/

      if(feof(fp))
          while( !kbhit() );

      strt+=(long)doffinc;
      fseek(fp,strt,SEEK_SET);

/*implement the FFT-display loop*/
      dataptr = revdata;
writpm(30, lp1, 12);
strt320(30);
tmsptr = (unsigned *)((lp2[0]<<1) | tmsaddr);
enable_p0;
      tmsfmt(dataptr,tmsptr,ptknt, gain);/* default ptknt = 128*/
    for (j=0; j<128; j++)
    colori[127-j] = (char)fftval[j];
```

```
    scrntop(color1, cycle++);

    if (cycle == 8) {
   cycle = 0;
   movetop();
     }

readdm(lp2[1], fftval, fftout);
ltmp = lp2;
lp2 = lp1;
lp1 = ltmp;
while((inp(IOBASE) & 0x08) == 0);
hlt320();

 }  /*end of user-control loop*/

}


/*********************************************************************/
/*********************************************************************/
/*SAVSCR.C
  savscr.c -- stores a section of data from the memory buffer that
              previously has been displayed on the screen.
              Data are either written to an output file which contains
              a header (allowing input to the Kay Sona-Graph
              Workstation), or are appended to a user-specified
              the file (non-Kay mode).
  note: code is present for the saving of color-coded data, but it is
        not enabled.
*/
/**********************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graph.h>
#include <sys\types.h>
#include <sys\stat.h>

#include "hsmos.def"

#include "fcntl.h"
#define O_RAW O_BINARY
#define enable_p1   wstcr( (rstcr()&STC_RUN) | STC_DPAGE1 )
#define disable_edm wstcr( rstcr()&STC_MEM_MASK )
#define enable_p0   wstcr( (rstcr()&STC_RUN) | STC_DPAGE0 )

unsigned int totms1();

char samples[8];
char opnmode[3];
char kaymode[] = "wb";
char etcmode[] = "ab";

/**********************************************************************/

savscr(pstrt,offstrt,offend,pageknt,step)
  unsigned int pstrt, offstrt, offend, step;
  int pageknt;
{
extern int fftval[], *lp1, *lp2, *ltmp, lcol, rcol, kay;
extern char fidid[], newfid[], color1[];
```

```
extern unsigned int *tmsptr;
extern long unsigned int tmsaddr;
extern FILE *stream;
 *
struct stat buf;
unsigned long int pageaddr, binknt, stand;
long int place;
unsigned int rawval;
unsigned int *newptr, outval, ofset, oldof;
int result, dtyp, txtcolor, savgain, i, j;

savgain = 0;
dtyp = 0;
result = -1;
binknt = 0;   /*for accruing number of words sent to saved Kay file*/

/*note: To implement filesaves of color-coded data or of reviewed data, dtyp
      must be added to the argument list. Then dtyp would equal 0 for dma
      input, 1 for file input, 2 to output color codes*/

/*get a filename from the user*/
  messages(5);

restart:
  gets(newfid);

 /*do we have a new filename, or just a <CR> ?*/
  if(strlen(newfid) > 0)
    {
       /*see if this file already exists*/

       if( stat(newfid,&buf) == 0 )
          {
           /*this file exists already -- set up for an append, or request
             a new filename if this is to be a Kay header file*/
           for( j=0 ; j<15 ; j++)
               newfid[j] = ' ';

           if( kay )
             {
              messages(10);
              goto restart;
              }
           else
             {
              if( messages(6) == 1 )
                 {
                   messages(5);
                   goto restart;
                  }
              }
           }

    }
  else
    {
     /*user hit <CR> -- no save is made*/
     messages(11);
     return(0);
     }

    /*Open the new data file*/
      if(dtyp < 2)
        {
```

```
   /*binary write or append of raw data*/

  if( kay )
      strcpy(opnmode,kaymode); /*Kay header file*/
  else
      strcpy(opnmode,etcmode); /*non-Kay; can be appended*/

  if((stream = fopen(newfid,opnmode)) == NULL){
      _clearscreen(_GCLEARSCREEN);
      _displaycursor(_GCURSORON);
       setscr();
      _setvideomode(_DEFAULTMODE);
      printf("Cannot open a new file. Your hard disk may be full.\n");
      clearerr(stream);
      exit(0);
      }
    }
 else
   /* append of colorcoded data*/
  if((stream = fopen(newfid,"a")) == NULL){
      _clearscreen(_GCLEARSCREEN);
      _displaycursor(_GCURSORON);
      setscr();
      _setvideomode(_DEFAULTMODE);
      printf("Cannot open a new file. Your hard disk may be full.\n");
      clearerr(stream);
      exit(0);
      }

/*redefine fidid*/
 strcpy(fidid,newfid);

messages(7);

/*write data to either old or new file*/
   /*save dma data*/
if(dtyp == 0)
  {
  if( kay )
    {
     kayhdr();
     }
  ofset = offstrt;
  oldof = 0;
  pageaddr = ((long)pstrt)<<28;

  /*loop once for each page, stopping on last page when offend is
    reached*/
  for( j=0 ; j <= pageknt ; j++)
    {
     /* do once at start of page to set ofset > 0 */
      newptr = (unsigned *)(ofset | pageaddr);
      rawval = *newptr;
      if( kay )
        rawval = rawval >> 4;

      if(putw(rawval,stream) == EOF){
        if(ferror(stream)){
          /*cannot write to file - the disk may be full*/
         _clearscreen(_GCLEARSCREEN);
         _displaycursor(_GCURSORON);
         setscr();
         _setvideomode(_DEFAULTMODE);
         printf("Cannot write to file. Your hard disk may be full.\n");
```

```
                clearerr(stream);
                exit(0);
                }
             }
            oldof = ofset;
            ofset += 2;
            ++binknt;

          while( ofset > oldof )
           {
            newptr = (unsigned *)(ofset | pageaddr);
            rawval = *newptr;
            if( kay )
                rawval = rawval >> 4;

            if(putw(rawval,stream) == EOF){
              if(ferror(stream)){
                /*cannot write to file - the disk may be full*/
                _clearscreen(_GCLEARSCREEN);
                _displaycursor(_GCURSORON);
                setscr();
                _setvideomode(_DEFAULTMODE);
                printf("Cannot write to file. Your hard disk may be full.\n");
                clearerr(stream);
                exit(0);
                }
             }
            oldof = ofset;
            ofset += 2;
            if( (j == pageknt) && ( (ofset+step) >= ofiend) )
                {
/*if( kay ), plug in binknt number of samples written*/
                if( kay )
                  {
                   rewind(stream);
                   place = 26;
                   stand = 10000;
                   while( binknt < stand )
                    {
                     ++place;
                     stand /= 10;
                     }
                   fseek( stream, place, SEEK_SET );
                   ultoa( binknt, samples, 10);
                   fputs( samples, stream);
                   }
                messages(8);
                fclose(stream);
                return(0);
                }
            ++binknt;
           }
         oldof = 0;
         ++pstrt;
         if(pstrt > 9) pstrt = 4;
         pageaddr = ((long)pstrt)<<28;
         }
       }

/****from here on down is disabled by dtyp being hardwired to 0 ****/
   /* if this is to be enabled, the code must be revised */

   else if (dtyp == 1)
/*save data input from a file*/
```

```
        return; /*. . . to function 'review', to save saved data*/

   else if (dtyp == 2)
      /*save the colorcodes of expanded fftvalues*/
      {
      outval = (long int)newptr;
      ofset = outval & 0xFFFF;
      for( i = 0 ; i < 480 ; i++)
          {
           writpm(30, lp1, 12);
           strt320(30);
           /*newptr = (unsigned *)(ofset | absaddr);*/
           tmsptr = (unsigned *)((lp2[0]<<1) | tmsaddr);
           enable_p0;
           totms1(newptr, tmsptr, 256, savgain);
           ofset += 136;
           for( j = 0 ; j < 128 ; j++)
              {
              color1[j] = (char)fftval[j];
              putc(color1[j],stream);
              }
           for( j = 0 ; j < 512 ; j++);  /*wait loop*/

           readdm(lp2[1], fftval, 256);
           ltmp = lp2;
           lp2 = lp1;
           lp1 = ltmp;
           while((inp(IOBASE) & 0x08) == 0);
           hlt320();
          }
      }

}


/*****************************************************************/
/*****************************************************************/
;SCREEN.ASM
;
; Draws the sonogram image to screen.
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)
;
;   note: 9/89 - Routines setscr, scrntop and scrnbot are functional. Other
;   code exists as hooks for future expansion.

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT
;setscr resets palette values to default colors
```

```
PUBLIC _setscr

_setscr PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di

        mov     ah, 0
        mov     al, 010h                ;set video mode 10h
        int     10h

        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_setscr ENDP

PUBLIC _scrntop

_scrntop  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     di, [bp+6]              ;pass offset of data buffer
        mov     ax, [bp+8]              ;pass segment to ds
        mov     ds, ax

        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     cx, [bp+10]
        mov     ax, 80h
        shr     ax, cl
        out     dx, al                  ;send data to mask register

;draw a pixel
        mov     bx, 60                  ;load with screen column number to write to
col1:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, ds:[di]             ;pixel color from input array
        mov     es:[bx], al             ;draw the pixel
        inc     di                      ;point to next input array element
        add     bx, 80                  ;point to pixel below
        cmp     bx, 10240               ;10240 OR 20480 for 256 pts (column bottom)
```

```
        jl      coll

finish:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_scrntop   ENDP

PUBLIC _scrnbot

_scrnbot  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     di, [bp+6]              ;pass offset of data buffer
        mov     ax, [bp+8]              ;pass segment to ds
        mov     ds, ax

        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     cx, [bp+10]
        mov     ax, 80h
        shr     ax, cl
        out     dx, al                  ;send data to mask register

;draw a pixel
        mov     bx, 60+10240            ;load with screen column number to write to
cola:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, ds:[di]             ;pixel color from input array
        mov     es:[bx], al             ;draw the pixel
        inc     di                      ;point to next input array element
        add     bx, 80                  ;point to pixel below
        cmp     bx, 20480               ;10240 OR 20480 for 256 pts
        jl      cola

bye:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret
```

```
_scrnbot   ENDP

PUBLIC _scrn1

_scrn1 PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     di, [bp+6]              ;pass offset of data buffer
        mov     ax, [bp+8]              ;pass segment to ds
        mov     ds, ax

        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     cx, [bp+10]
        mov     ax, 80h
        shr     ax, cl
        out     dx, al                  ;send data to mask register

;draw a pixel
        mov     bx, 60                  ;load with screen column number to write to
colb:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, ds:[di]             ;pixel color from input array
        mov     es:[bx], al             ;draw the pixel
        inc     di                      ;point to next input array element
        add     bx, 80                  ;point to pixel below
        cmp     bx, 5120                ;10240 OR 20480 for 256 pts
        jl      colb

end1:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_scrn1   ENDP

PUBLIC _scrn2

_scrn2 PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
```

```
        push    di
        push    ds

        mov     di, [bp+6]          ;pass offset of data buffer
        mov     ax, [bp+8]          ;pass segment to ds
        mov     ds, ax

        mov     dx, 3ceh            ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2               ;video mode 2
        out     dx, al
        mov     ax, 0a019h          ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh            ;point to address register
        mov     al, 8               ;bit mask register
        out     dx, al              ;address the register
        inc     dx                  ;point to data register
        mov     cx, [bp+10]
        mov     ax, 80h
        shr     ax, cl
        out     dx, al              ;send data to mask register

;draw a pixel
        mov     bx, 60+5120         ;load with screen column number to write to
colc:   mov     al,es:[bx]          ;fill the latch registers
        mov     al, ds:[di]         ;pixel color from input array
        mov     es:[bx], al         ;draw the pixel
        inc     di                  ;point to next input array element
        add     bx, 80              ;point to pixel below
        cmp     bx, 10240           ;10240 OR 20480 for 256 pts
        jl      colc

end2:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_scrn2  ENDP

PUBLIC _scrn3

_scrn3 PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     di, [bp+6]          ;pass offset of data buffer
        mov     ax, [bp+8]          ;pass segment to ds
        mov     ds, ax

        mov     dx, 3ceh            ;set video write mode 2
        mov     al, 5
        out     dx, al
```

```
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     cx, [bp+10]
        mov     ax, 80h
        shr     ax, cl
        out     dx, al                  ;send data to mask register

;draw a pixel
        mov     bx, 60+10240            ;load with screen column number to write to
cold:   mov     al,es:[bx]              ;fill the latch registers
        mov     al, ds:[di]             ;pixel color from input array
        mov     es:[bx], al             ;draw the pixel
        inc     di                      ;point to next input array element
        add     bx, 80                  ;point to pixel below
        cmp     bx, 15360               ;10240 OR 20480 for 256 pts
        jl      cold

end3:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_scrn3  ENDP

PUBLIC _scrn4

_scrn4 PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds

        mov     di, [bp+6]              ;pass offset of data buffer
        mov     ax, [bp+8]              ;pass segment to ds
        mov     ds, ax

        mov     dx, 3ceh                ;set video write mode 2
        mov     al, 5
        out     dx, al
        inc     dx
        mov     al, 2                   ;video mode 2
        out     dx, al
        mov     ax, 0a019h              ;point to top left screen corner
        mov     es, ax

;set up bit mask register
        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
```

```
        inc     dx                          ;point to data register
        mov     cx, [bp+10]
        mov     ax, 80h
        shr     ax, cl
        out     dx, al                      ;send data to mask register

;draw a pixel
        mov     bx, 60+15360                ;load with screen column number to write to
cole:   mov     al,es:[bx]                  ;fill the latch registers
        mov     al, ds:[di]                 ;pixel color from input array
        mov     es:[bx], al                 ;draw the pixel
        inc     di                          ;point to next input array element
        add     bx, 80                      ;point to pixel below
        cmp     bx, 20480                   ;10240 OR 20480 for 256 pts
        jl      cole

end4:
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_scrn4   ENDP

_TEXT ENDS

END


/*********************************************************************/
/*********************************************************************/
/*SHOWDAT.C
  showdat.c -- Effects retrieval of data previously stored in multiple
  pages of memory by the DMA, recomputes the FFTs, and displays
  one line each time it is called.
*/
/**********************************************************************/

#include <stdio.h>
#include "hsmos.def"

#define O_RAW O_BINARY
#define enable_p1   wstcr( (rstcr()&STC_RUN) | STC_DPAGE1 )
#define disable_edm wstcr( rstcr()&STC_MEM_MASK )
#define enable_p0   wstcr( (rstcr()&STC_RUN) | STC_DPAGE0 )

unsigned int scrntop();
unsigned int scrnbot();
unsigned int totms1();
unsigned int totms2();
unsigned int (*move)();
unsigned int (*tmsfmt)();

/**********************************************************************/

showdat(disp,tock,decem)
  int disp;  /*display option set in cmdopt: 1 = 4-channel, 2 = 2-channel,
              >2 = 1-channel*/
  int tock;  /*set to 1 or 0 in getmem for one-second time ticks*/
  int decem; /*set to 1 or 0 in getmem for tenth-of-a-second ticks*/
 {
   extern int cycle, ptknt, fftout, fftval[], *lp1, *lp2, *ltmp;
```

```
extern int lcol, rcol;
extern int gain;
extern char color1[], color2[];
extern unsigned int *dataptr, *tmsptr, doffset, envelope;
extern long unsigned int tmsaddr, showaddr;

int i,j,k,n, exgain;
static int tenknt;
exgain = gain;
envelope = 0;


    writpm(30, lp1, 12);
    strt320(30);
    dataptr =(unsigned *)(doffset | showaddr);
    tmsptr = (unsigned *)((lp2[0]<<1) | tmsaddr);
    enable_p0;
    tmsfmt(dataptr,tmsptr,ptknt, exgain);

    /*find the signal amplitude*/
    for( i=0 ; i<ptknt ; i++ )
       if( envelope < dataptr[i] ) envelope = dataptr[i];
    envelope = (envelope - 0x7fff) >> 8;


    if(disp == 2)
      {
      k = 0;
      for( j = 0 ; j < 128 ; j++)
         {
         color1[127-j] = (char)fftval[k++];
         color2[127-j] = (char)fftval[k++];
         }

     /*make the time ticks*/
      if( tock )
         {
           /*make a longer line every ten seconds*/
           ++tenknt;
           if( !(tenknt%10) )
             {
              tenknt = 0;
              for( j = 120 ; j < 128 ; j++)
                 color1[127-j] = 63;
              }
           else
              for( j = 124 ; j < 128 ; j++)
                 color1[127-j] = 63;
         }

      if( decm )
        {
              for( j = 126 ; j < 128 ; j++)
                 color1[127-j] = 63;
         }

      color1[127] = 63; /*horizontal line across display area*/

      scrntop(color1,cycle);
      scrnbot(color2,cycle);
      cycle++;
      }
   else if (disp > 2)
      {
```

```
        for (j=0; j<128; j++)
            color1[127-j] = (char)fftval[j];

        /*make the time ticks*/
        if( tock )
           {
             /*make a longer line every ten seconds*/
             ++tenknt;
             if( !(tenknt%10) )
               {
                tenknt = 0;
                for( j = 120 ; j < 128 ; j++)
                   color1[127-j] = 63;
                }
             else
                for( j = 124 ; j < 128 ; j++)
                   color1[127-j] = 63;
           }
        if( decem )
          {
                for( j = 126 ; j < 128 ; j++)
                   color1[127-j] = 63;
           }
        color1[127] = 63; /*horizontal line across display area*/

        scrntop(color1, cycle);
        putmag(&envelope,cycle);
        cycle++;
        }

       if (cycle >= 8) {
          cycle = 0;
          movefull(); /* a wide waveform for all memory displays*/
        }

     readdm(lp2[1], fftval, fftout);
     ltmp = lp2;
     lp2 = lp1;
     lp1 = ltmp;
     while((inp(IOBASE) & 0x08) == 0);
     hlt320();

  return;
}


/**********************************************************************/
/**********************************************************************/
;TOTMS.ASM
;totms.asm -- moves a buffer from the DMA buffer to TMS data memory,
; formats and scales the data buffer on the fly.
; Call with:
; address of input buffer
; address of TMS   buffer
; number of words to transfer
; scaling coefficient (gain)
;
; source segment    - es
; source offset     - si
; destination offset - di
;
; Negative gain is allowed.
;****************************************************
_TEXT  SEGMENT BYTE PUBLIC 'CODE'
```

```
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP

_TEXT   SEGMENT

;****************************************************
;This version of totms is for displaying and saving a single channel.

PUBLIC _totms1

_totms1  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds
        push    es

;called with (*source, *dest, numwords, gain)

        mov     si, [bp+6]              ;pass source offset
        mov     ax, [bp+8]              ;pass source segment
        mov     es, ax                  ;to es

        mov     di, [bp+10]             ;pass destination offset
        mov     ax, [bp+12]             ;pass destination segment
        mov     ds, ax                  ;to ds

        mov     dx, [bp+14]             ;get number of words to move
        mov     cx, [bp+16]             ;get gain

        mov     bx, 07fffh

begin:  cmp     cx,0                    ;see if gain is < 0
        jl      less

loop1:  mov     ax, es:[si]             ;get source word
        sub     ax, bx                  ;unipolar -> bipolar
        shl     ax, cl                  ;adjust the gain
        mov     [di], ax                ;put it to destination (real buffer)
        add     di, 2                   ;increment destination address
        mov     ax, es:[si]
        sub     ax, bx
        shl     ax, cl
        mov     [di], ax
        add     di, 2                   ;GET READY FOR NEXT LOOP -
        add     si, 2

        dec     dx
        jnz     loop1
        jmp     bye
```

```
less:   neg     cx                      ;change sign from neg to pos

loop0:  mov     ax, es:[si]             ;get source word
        sub     ax, bx                  ;unipolar -> bipolar
        sar     ax, cl                  ;adjust the gain
        mov     [di], ax                ;put it to destination (real buffer)
        add     di, 2                   ;increment destination address
        mov     ax, es:[si]
        sub     ax, bx
        sar     ax, cl
        mov     [di], ax
        add     di, 2                   ;GET READY FOR NEXT LOOP -
        add     si, 2

        dec     dx
        jnz     loop0
        jmp     bye

bye:    pop     es
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_totms1 ENDP

;**************************************************

;This version is used for displaying 2 channels
PUBLIC _totms2

_totms2  PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di
        push    ds
        push    es

;called with (*source, *dest, numwords, gain)

        mov     si, [bp+6]              ;pass source offset
        mov     ax, [bp+8]              ;pass source segment
        mov     es, ax                  ;to es

        mov     di, [bp+10]             ;pass destination offset
        mov     ax, [bp+12]             ;pass destination segment
        mov     ds, ax                  ;to ds

        mov     dx, [bp+14]             ;get number of words to move
        mov     cx, [bp+16]             ;get gain

        mov     bx, 07fffh

strt:   cmp     cx,0                    ;see if gain is < 0
        jl      minus

loop2:
        mov     ax, es:[si]             ;get source word
```

```
        sub     ax, bx                  ;unipolar -> bipolar
        shl     ax, cl                  ;adjust the gain
        mov     [di], ax                ;put it to destination (real buffer)
        add     si, 2
        add     di, 2
        mov     ax, es:[si]
        sub     ax, bx
        shl     ax, cl
        mov     [di], ax
        add     di, 2
        add     si,2


        dec     dx
        jnz     loop2
        jmp     finish

minus:  neg     cx                      ;a negative gain is allowed
loop2a:  mov    ax, es:[si]             ;get source word
        sub     ax, bx                  ;unipolar -> bipolar
        sar     ax, cl                  ;adjust the gain
        mov     [di], ax                ;put it to destination (real buffer)
        add     di, 2
        add     si, 2                   ;increment location in segment source
        mov     ax, es:[si]
        sub     ax, bx
        sar     ax, cl
        mov     [di], ax
        add     di, 2
        add     si, 2                   ;increment location in segment source

        dec     dx
        jnz     loop2a

finish: pop     es
        pop     ds
        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_totms2 ENDP

_TEXT ENDS

END


/**********************************************************************/
/**********************************************************************/
;TXTPREP.ASM
;
;Restores bit map area and bit address register to allow display of
;MSC Graphics text on screen.
;
;Write directly to the EGA video RAM. This routine assumes the video driver
;is IBM compatible and supports EGA mode 10H ( 640x350, 16 colors)
;

_TEXT  SEGMENT BYTE PUBLIC 'CODE'
_TEXT  ENDS

_DATA   SEGMENT WORD PUBLIC 'DATA'
```

```
_DATA  ENDS

CONST  SEGMENT  WORD  PUBLIC 'CONST'
CONST  ENDS

_BSS   SEGMENT  WORD PUBLIC  'BSS'
_BSS   ENDS

DGROUP  GROUP  CONST,   _BSS,   _DATA
   ASSUME  CS:_TEXT, DS:DGROUP, SS:DGROUP, ES:DGROUP


_TEXT   SEGMENT

PUBLIC _txtprep

_txtprep PROC FAR

        push    bp
        mov     bp,sp
        push    ss
        push    si
        push    di

        mov     dx, 3ceh                ;set write mode 1
        mov     al, 5                   ;index register 5
        out     dx, al                  ;send the index
        inc     dx                      ;point to mode register
        mov     al, 0                   ;choose mode 0
        out     dx, al                  ;set the mode

        mov     dx, 3ceh                ;point to address register
        mov     al, 8                   ;bit mask register
        out     dx, al                  ;address the register
        inc     dx                      ;point to data register
        mov     al, 0ffh
        out     dx, al                  ;send data to mask register

        pop     di
        pop     si
        pop     ss
        pop     bp
        ret

_txtprep ENDP

_TEXT  ENDS

END

/**********************************************************************/
/**********************************************************************/
```

DOCUMENT LIBRARY

January 17, 1990

Distribution List for Technical Report Exchange

Attn: Stella Sanchez-Wade
Documents Section
Scripps Institution of Oceanography
Library, Mail Code C-075C
La Jolla, CA 92093

Hancock Library of Biology &
Oceanography
Alan Hancock Laboratory
University of Southern California
University Park
Los Angeles, CA 90089-0371

Gifts & Exchanges
Library
Bedford Institute of Oceanography
P.O. Box 1006
Dartmouth, NS, B2Y 4A2, CANADA

Office of the International
Ice Patrol
c/o Coast Guard R & D Center
Avery Point
Groton, CT 06340

NOAA/EDIS Miami Library Center
4301 Rickenbacker Causeway
Miami, FL 33149

Library
Skidaway Institute of Oceanography
P.O. Box 13687
Savannah, GA 31416

Institute of Geophysics
University of Hawaii
Library Room 252
2525 Correa Road
Honolulu, HI 96822

Marine Resources Information Center
Building E38-320
MIT
Cambridge, MA 02139

Library
Lamont-Doherty Geological
Observatory
Columbia University
Palisades, NY 10964

Library
Serials Department
Oregon State University
Corvallis, OR 97331

Pell Marine Science Library
University of Rhode Island
Narragansett Bay Campus
Narragansett, RI 02882

Working Collection
Texas A&M University
Dept. of Oceanography
College Station, TX 77843

Library
Virginia Institute of Marine Science
Gloucester Point, VA 23062

Fisheries-Oceanography Library
151 Oceanography Teaching Bldg.
University of Washington
Seattle, WA 98195

Library
R.S.M.A.S.
University of Miami
4600 Rickenbacker Causeway
Miami, FL 33149

Maury Oceanographic Library
Naval Oceanographic Office
Stennis Space Center
NSTL, MS 39522-5001

Marine Sciences Collection
Mayaguez Campus Library
University of Puerto Rico
Mayagues, Puerto Rico 00708

Library
Institute of Oceanographic Sciences
Deacon Laboratory
Wormley, Godalming
Surrey GU8 5UB
UNITED KINGDOM

The Librarian
CSIRO Marine Laboratories
G.P.O. Box 1538
Hobart, Tasmania
AUSTRALIA 7001

Library
Proudman Oceanographic Laboratory
Bidston Observatory
Birkenhead
Merseyside L43 7 RA
UNITED KINGDOM

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO. WHOI-90-22	2.	3. Recipient's Accession No.
4. Title and Subtitle VOICE – A Spectrogram Computer Display Package			5. Report Date June, 1990
			6.
7. Author(s) Ann Martin, Josko A. Catipovic, Kurt Fristrup, and Peter L. Tyack			8. Performing Organization Rept. No. WHOI 90-22
9. Performing Organization Name and Address The Woods Hole Oceanographic Institution Woods Hole, Massachusetts 02543			10. Project/Task/Work Unit No.
			11. Contract(C) or Grant(G) No. (C) N00014-88-K-0273 (G) N00014-87-K-0236 1 R29 NS25290
12. Sponsoring Organization Name and Address Funding was provided by the Office of Naval Research, the National Institutes of Health, and the Andrew W. Mellon Foundation.			13. Type of Report & Period Covered Technical Report
			14.

15. Supplementary Notes

This report should be cited as: Woods Hole Oceanog. Inst. Tech. Rept., WHOI-90-22.

16. Abstract (Limit: 200 words)

A real-time spectrogram instrument has been developed to provide an inexpensive and field-portable instrument for the analysis of animal sounds. The instrument integrates a computer graphics display package with a PC-AT computer equipped with an A/D board and a digital signal processing board. It provides a real-time spectrogram display of frequencies up to 50kHz in a variety of modes: a running display, a signal halted on screen, successive expanded views of the signal. The signal amplitude may also be displayed. Portions of the scrolled data may be saved to disk file for future viewing, or as part of a database collection. The screen display may be manipulated to adapt to special needs. Program source listings are included in the text.

17. Document Analysis a. Descriptors

1. spectrogram
2. digital signal processing
3. bioacoustic analysis
4. computer software

b. Identifiers/Open-Ended Terms

c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Approved for publication; distribution unlimited.	UNCLASSIFIED	96
	20. Security Class (This Page)	22. Price

(See ANSI-Z39.18) See Instructions on Reverse OPTIONAL FORM 272 (4-77)
(Formerly NTIS-35)
Department of Commerce